MIT CSAIL FastCode Seminar 10/26/2020: Professor Alex Pothen

1 00:00:06.299 --> 00:00:15.570
Julian Shun: Good afternoon, everyone. Welcome to the FastCode seminar today. So I'm very happy to have Alex Pothen as our speaker Alex is a professor of Computer Science.

2 00:00:15.960 --> 00:00:28.020
Julian Shun: At Purdue University, and he received his undergraduate degree from IIT Delhi and his PhD from Cornell, Alex. His research interests include combinatorial scientific computing

3 00:00:28.470 --> 00:00:34.320
Julian Shun: parallel computing and bioinformatics and Alex let the formation of the Communist oil scientific computing

4 00:00:34.800 --> 00:00:45.780
Julian Shun: Community within the society for Industrial and Applied Mathematics and He currently serves as the founding chair of the same activity group on applied in computational discrete algorithms.

5 00:00:46.200 --> 00:00:57.300
Julian Shun: Alex has mentored 20 PhD students postdoctoral scientists and research faculty who have gone on to successful careers at major universities do you labs and industry.

6 00:00:58.470 --> 00:01:11.340
Julian Shun: And Alex serves on the editorial board of the journal ACM as well as the Science Review and Alex is also a fellow of the society of Industrial and Applied Mathematics.

7 00:01:11.820 --> 00:01:19.410
Julian Shun: And today, Alex is going to talk to us about his work on approximate parallel graph algorithms. So I'll turn it over to you, Alex.

8 00:01:20.550 --> 00:01:29.310
Alex Pothen: Thank you. Julian, I trust everybody can hear me. I apologize. You can't see my face. That's because I'm presenting using my iPad and

9 00:01:29.700 --> 00:01:32.790
Alex Pothen: A program called Notability in which I have imported a PDF file.

10 00:01:33.090 --> 00:01:36.060
Alex Pothen: So unfortunately, I think. I don't think there's a way to keep my

11 00:01:37.110 --> 00:01:47.910

Alex Pothen: Camera on why that I'm screen sharing on that is not on this, on this app that for a second. Tell so so sorry about that. So I'm just a disembodied voice.

12 00:01:48.720 --> 00:02:04.290
Alex Pothen: From somewhere far away, but I hope that you will enjoy listening to this lecture. And please feel free to ask questions as we go along. And again, Julian, feel free to stop me at any point if there's a question coming through on the chat window and

13 00:02:05.040 --> 00:02:06.270
Alex Pothen: Ask. Sure. Okay.

14 00:02:06.630 --> 00:02:15.060
Alex Pothen: Alright, so, so as as Julian said, I'm going to talk about designing approximation algorithms as a paradigm for designing Tallulah

15 00:02:16.080 --> 00:02:25.560
Alex Pothen: So we have many different paradigms that we have for designing parallel for them. So for example, if it's data Paolo, you might want to petition the data or petition the graph.

16 00:02:25.920 --> 00:02:40.200
Alex Pothen: And then do computations on the sub graph and so on. And there are others like divide and conquer and pointed jumping and Delta stepping and that's all I wanted to today, talk about approximation algorithms themselves as a paradigm for designing

17 00:02:42.120 --> 00:02:52.920
Alex Pothen: And I want to motivate this by by showing you a result initially. So in this problem is called vertex, where it matching, which we will look at in a couple of slides.

18 00:02:53.760 --> 00:02:59.550
Alex Pothen: What I'm doing is I'm comparing an exact algorithm with a two thirds approximation now.

19 00:03:00.480 --> 00:03:08.160
Alex Pothen: So the two thirds approximation algorithm says I can't give you the exact solution, but I'll give you something that's within a factor of two thirds in the worst case, okay.

20 00:03:08.670 --> 00:03:17.220
Alex Pothen: And what we're doing is comparing against six problems and these have millions of vertices and say 10s of millions of edges, let's say,

21 00:03:17.790 --> 00:03:32.280

Alex Pothen: And you can see the exact algorithm takes typically something like, you know, an hour or so or slightly less than that. Right. And the approximation algorithm, which doesn't solve the problem exactly gives you a solution within 10 seconds.

22 00:03:33.630 --> 00:03:41.880
Alex Pothen: And although the approximation guarantees two thirds in practice, you actually get 99%, you know, or 98% or better.

23 00:03:43.260 --> 00:03:49.380
Alex Pothen: Okay, so here's this last problem that's interesting because this is not a very large problem has problems good these days but massive crafts.

24 00:03:50.010 --> 00:03:56.520
Alex Pothen: But still, because of the structure of this problem. The exact algorithm, you know, will not terminate, even in hundred dollars.

25 00:03:57.240 --> 00:04:05.490
Alex Pothen: Okay, it's pulling on the time complexity. It's still will not terminate in 100 hours, whereas the exact the approximation algorithm runs in under a minute.

26 00:04:06.120 --> 00:04:16.020
Alex Pothen: And of course because exact algorithm does not terminate. I don't know how close to to optimal. This is except for the fact that I have this guarantee that says I'm at least two thirds optimal, you know,

27 00:04:16.560 --> 00:04:34.650
Alex Pothen: A factor of two thirds of the optimal. So let's take a look at at at what you know. So first let's define some terminology and then we'll talk. Take a look at the problems that you're looking at so exact algorithm support solve the problem that's given to you and compute an exact solution.

28 00:04:36.180 --> 00:04:41.610
Alex Pothen: They may have a normal time complexity or even the problems are intractable than they might have expansion complexity.

29 00:04:42.390 --> 00:04:52.650
Alex Pothen: But even if you have problems like matching that we will look at today that have been on the time complexity. They can be quite slough a massive grass, grass with, you know, billions of edges.

30 00:04:53.580 --> 00:05:07.110

Alex Pothen: And often these algorithms are fairly sophisticated and there's not much concurrency and they're not easy to implement, even on serial machines, as many of you who have tried to implement some fairly challenging crap out with them can attest.

31 00:05:08.220 --> 00:05:11.790
Alex Pothen: So instead of exact algorithms will go to approximation on us.

32 00:05:12.510 --> 00:05:24.180
Alex Pothen: So here we give up on finding the exact solution. But he said, Okay, well, just wanted to prove something about the solution that you have came at it. Some factors and some factor of the optimum the worst case.

33 00:05:25.020 --> 00:05:30.600
Alex Pothen: Factor that you can obtain for all over all problems because the approximation ratio of the of the algorithm.

34 00:05:31.650 --> 00:05:42.210
Alex Pothen: But as you saw in my previous slide these solutions can be nearly optimal in practice because the worst case approximation ratio is not what you actually get, you know, in an average case sense

35 00:05:43.440 --> 00:05:52.320
Alex Pothen: And these algorithms of simpler therefore can be implemented efficiently can also be designed to have concurrency until we can also implement them efficiently on power ships.

36 00:05:53.550 --> 00:06:04.740
Alex Pothen: So I want to make a distinction between approximation algorithms and heuristic algorithms. So heuristics are where there is no approximation ratio that they can prove what that is is known

37 00:06:05.460 --> 00:06:11.460
Alex Pothen: In many cases, it may be possible to prove that a good approximation algorithm does not exist because we have

38 00:06:11.760 --> 00:06:22.080
Alex Pothen: You know, some resolved that says, unless basically envy the such and such an algorithm know such and such a problem cannot be approximated within such in such a factor and someone, and there are many results of that nature.

39 00:06:24.150 --> 00:06:34.800
Alex Pothen: So let's now look at the matching problem itself. So hopefully many of you have seen matching in a in a graduate class in in in in algorithms.

40 00:06:35.670 --> 00:06:47.910
Alex Pothen: But any case give ghosts. So a matching is a set of vertex disjointed edges. So this means that we choose at most one edge incident on each works. So the gold edges here, the three of them.

41 00:06:48.240 --> 00:06:55.200
Alex Pothen: They form a matching because there's at most one gold edge incident on each box. Okay. There's none on this, but it says at most one

42 00:06:56.430 --> 00:07:06.840
Alex Pothen: So now we can ask. So if I give you this problem and ask you to find a matching, you could say, well, what do you want me to do, Joe. Should I maximize the cardinality. In this case it would be three.

43 00:07:07.260 --> 00:07:20.790
Alex Pothen: When should I put weight on the edges. And then, you know, add the some the weights of the matching edges. And so, and then maximize that some so we'll call the maximum edge rated matching problem.

44 00:07:22.410 --> 00:07:27.870
Alex Pothen: We could in fact put weight on the vertices. And then, and then

45 00:07:28.770 --> 00:07:37.860
Alex Pothen: Sum up the weeks of the endpoints of the matching edges so that will be a vertex weighted maximum vertex credit matching problem. So this is a problem that I'm looking at first.

46 00:07:38.190 --> 00:07:48.960
Alex Pothen: Then I'll look at this problem called be matching where instead of having this constrained there's at most one edge, we could say, well, there's that most some BV edges. So, for example, do if you'd like.

47 00:07:49.710 --> 00:07:53.730
Alex Pothen: But we could choose be the to be dependent on the itself, you know, and so on.

48 00:07:54.120 --> 00:08:05.700
Alex Pothen: Clean. It's got to be less than the degree of the vertex, but we could choose different that is to be and then again we could ask for a be managing all of these, these problems, you know, also have very nice applications. I'm looking at that as well. In the stock.

49 00:08:07.320 --> 00:08:15.150

Alex Pothen: So here's a quick sort of overview of where things are in this in this area when it comes to maximum matching problems.

50 00:08:15.540 --> 00:08:23.100
Alex Pothen: So if you look at cardinality. There is something like a point 866 approximation algorithm that was designed in the last few years.

51 00:08:23.940 --> 00:08:31.320
Alex Pothen: For estimated matching, you can get arbitrarily close to one. There's a one minus excellent approximation algorithm that dates back about six years to sell.

52 00:08:31.890 --> 00:08:43.560
Alex Pothen: Will I will discuss this algorithm, but I'll show you some results from this album, there's a two thirds minus excellent approximation algorithm. Unfortunately, these algorithms are fairly sophisticated and very hard to implement in parallel.

53 00:08:44.790 --> 00:08:52.320
Alex Pothen: But, like, one can implement many half approximation algorithm. So there are actually, at least you know 10 or so.

54 00:08:52.770 --> 00:09:00.930
Alex Pothen: Different half approximation algorithms for this problem. I'm going to talk about one of them you know by former student and and a colleague

55 00:09:01.590 --> 00:09:12.570
Alex Pothen: That was designed a few years ago, and then I'll talk about the work that's rated matching problem, which is what I'll go to next. So you can. There also you can get arbitrarily close to one, at least within some

56 00:09:13.530 --> 00:09:19.290
Alex Pothen: parameter k which is the length of a search path. And I'll talk about that in a bit. And, and then

57 00:09:19.950 --> 00:09:30.720
Alex Pothen: You can also look at edge weighted be matching. And there we have a half approximation algorithm that actually derives from this graduated matching algorithm of man and how the funnel works. So I'll talk about both of these. Let's talk

58 00:09:31.980 --> 00:09:39.390
Alex Pothen: And and some of these algorithms can be implemented in parallel, some of them cannot and so on. So, you know, we'll see them.

59 00:09:39.990 --> 00:09:44.280
Alex Pothen: Alright, so now for the first problem, which is a card maximum vertex, where the matching problem.

60 00:09:44.880 --> 00:09:59.610
Alex Pothen: And so in this case. Remember the weights around the vertices and Rob, you're asked to do is to find a matching and weight of a matching. This is some of the weeks of the endpoints of the matching edges and we want to maximize that something that's what we want to do.

61 00:10:00.720 --> 00:10:06.150
Alex Pothen: So one of the first questions I like to answer. When I look back, I'm giving a problem like this to think about

62 00:10:06.510 --> 00:10:17.610
Alex Pothen: Is to ask, you know, what is the underlying cause editorial structure. He said some beautiful communitarian structure underlying the problem that is helpful in solving this problem at least viewing this problem in a casual way.

63 00:10:18.390 --> 00:10:24.270
Alex Pothen: And the answer is yes, there is in fact a structure called a Metroid that color matching Metroid, that was

64 00:10:24.930 --> 00:10:29.370
Alex Pothen: Discussed very early BY JACK Edmonds and re flickers and in 1965

65 00:10:29.910 --> 00:10:34.290
Alex Pothen: Unfortunately, if you look at a book on matrix theory, you're not likely to find a good discussion.

66 00:10:34.560 --> 00:10:42.810
Alex Pothen: Of matching Metroid so very few books actually discuss them. So there's not a lot that has been done in this problem, but there is this beautiful matrix that we can look at

67 00:10:43.680 --> 00:11:01.650
Alex Pothen: And remind you that a Metroid is a collection is a pair of have a good set of vertices and some a collection of subsets of vertices that will call independent subsets and what makes us you know vertex upset independent well it's independent if the vertices in that set.

68 00:11:03.030 --> 00:11:15.270
Alex Pothen: Are you know contained among the endpoints, have some matching. So choose any matching doesn't have to be maximum any cardinality. And then you ask yourself, look at those endpoints.

69 00:11:15.810 --> 00:11:24.720
Alex Pothen: And then is i a subset of some of some such matching endpoints of some such matching. Okay then. Then you'll say it's independent

70 00:11:26.310 --> 00:11:37.020
Alex Pothen: The nice thing about having a matrix structure is that there is a greedy algorithm that solves the problem optimally, so you don't have to do anything more than to just look at the greedy paradigm for designing an hour.

71 00:11:37.920 --> 00:11:44.160
Alex Pothen: And so, turns out that the matrix greedy algorithm, which I will explain in just a little bit computing up to a match.

72 00:11:44.700 --> 00:11:51.510
Alex Pothen: Okay. And furthermore, if you are willing to look at problems that are even more general. So you're not looking at some of

73 00:11:51.870 --> 00:12:04.920
Alex Pothen: The weights of the edges as your objective function, but you're looking at something like the square root of the size of the weight or something like that, right, then those are called some major objective functions and there. It turns out to be a half half approximation.

74 00:12:07.080 --> 00:12:14.910
Alex Pothen: Okay, so what does it make you a good yeah with them simply says take the vertices, put it sort them, put them in not increasing order weights.

75 00:12:15.300 --> 00:12:22.380
Alex Pothen: And then choose the vertex to match and then you know you can you see if you can find a matching that includes that Botox.

76 00:12:22.740 --> 00:12:30.300
Alex Pothen: If it does, then you okay that goes into the into the set the independent said that you have, if not you rejected and you continue

77 00:12:30.720 --> 00:12:43.200
Alex Pothen: Right so clearly searching for, you know, a matching which which we will discuss in terms of search box in the next couple of slides is going to be linear in the number of the linear in the number of edges.

78 00:12:44.730 --> 00:12:54.150

Alex Pothen: And and there are in such variety. So it's actually an order in and time out for them. It turns out that you can implement it a little faster, but particularly particularly helpful.

79 00:12:54.660 --> 00:13:02.430
Alex Pothen: Unfortunately, this is this order nm time is phenomenal. But it's still too slow for grabs, with to billions of vertices and edges. Okay.

80 00:13:03.660 --> 00:13:09.930
Alex Pothen: And so here you know so those of you know something about matching know that you look for certain paths called mounting paths or something.

81 00:13:10.710 --> 00:13:16.530
Alex Pothen: But by restricting the length of such search paths you can in fact get two thirds approximation algorithm.

82 00:13:17.160 --> 00:13:24.630
Alex Pothen: But with near linear time complexity. So instead of the factor and you have something like the maximum degree. So if you have a degree boundary graph.

83 00:13:24.990 --> 00:13:34.500
Alex Pothen: Then it's actually, you know, linear time linear time algorithm. Okay. Unfortunately, although we have this beautiful theory to undergird our work.

84 00:13:35.370 --> 00:13:40.890
Alex Pothen: These algorithms don't have much concurrency, because we do have to process the vertices in a particular order.

85 00:13:41.250 --> 00:13:54.570
Alex Pothen: But you can that you can in fact cheat a little bit, you can kind of lazy really algorithms and so on. So, that I think you can do but but it turns out that there is a better approach and then that's the approach that I want to turn to in the next few minutes or so.

86 00:13:56.460 --> 00:14:08.040
Alex Pothen: So here I need a couple of concepts I knew the concept of romantic path which I had alluded to, just a little bit, which is the the framework underlying most matching algorithms.

87 00:14:08.460 --> 00:14:16.950
Alex Pothen: Right, so you have a matching in the graph startling empty matching, let's say, and then they'll say that a path is alternating if it's

88 00:14:18.120 --> 00:14:28.800

Alex Pothen: Just belong is doesn't belong to the matching belongs to the matching and belong. It doesn't belong to the matching. So, this is an alternating path right so the matching and non matching just sort of alternate

89 00:14:29.760 --> 00:14:38.430
Alex Pothen: These in this case will call us and augmenting path if there are, you know, there's one more non matching edge than there is a matching edge.

90 00:14:38.700 --> 00:14:46.860
Alex Pothen: Because then by swapping matching and non matching. I just asked here we can increase outside the cardinality of the matching to to

91 00:14:47.160 --> 00:14:58.650
Alex Pothen: You know, instead of one here, right. And then we have unmatched that particular edge. Okay, so in this case we have an odd length alternating path that begins and ends with an unmatched vortex.

92 00:14:59.040 --> 00:15:08.970
Alex Pothen: And by simply swapping matching and non matching edges be increased the cardinality. So this is essentially the basic framework that is used for a lot of matching algorithms.

93 00:15:09.870 --> 00:15:20.910
Alex Pothen: Even when there are weeks, but I need one more concept and that concept is called awake increasing path. So this is now also an alternating path. So the matching and non matching. I just thought

94 00:15:21.960 --> 00:15:33.510
Alex Pothen: But for now we have weight on the vertices in this case. And I have a this is an even length alternating pencil exactly an equal number of matching edges and non matching edges.

95 00:15:34.050 --> 00:15:44.070
Alex Pothen: And now I have the non the unmatched end point is heavier than the match endpoint. So again, by swapping matching and non matching. I just as I haven't gotten here.

96 00:15:45.540 --> 00:15:54.450
Alex Pothen: I can know increase the week of the matching because I lose three because I I make an unmatched. But then I gained 10 because I've matched this edge.

97 00:15:54.960 --> 00:16:03.930

Alex Pothen: So again, by using this alternating pattern swapping matching and non matching edges I can increase the weight. So we'll call this a wage increasing path or an increasing path for short.

98 00:16:04.500 --> 00:16:10.320
Alex Pothen: So these are the two paths that we need. We need the concept of an automatic back then, we need the concept of awake increasing death.

99 00:16:10.830 --> 00:16:18.150
Alex Pothen: So once we have these two, we can generalize a little bit just to make it easier for us to think about us.

100 00:16:18.630 --> 00:16:26.400
Alex Pothen: So I will say, and augmenting path or a weight increasing path, but most keen on matching edges you secure mentation

101 00:16:26.910 --> 00:16:37.860
Alex Pothen: Okay. So for example, this is a one augmentation because it's a it's an authentic path with exactly one non matching edge. And so by matching it I can increase the size of the matching. So, not putting path.

102 00:16:38.430 --> 00:16:48.000
Alex Pothen: Here's a way to increasing path of link to their, you know, one non matching edge in one matching edge. But if the weights are appropriate. So I can buy swapping the

103 00:16:48.270 --> 00:16:56.400
Alex Pothen: You know, matching and non matching edges I could increase the weight. So this is, in fact, also a one augmentation because it's got at most one non matching edge.

104 00:16:57.630 --> 00:17:01.200
Alex Pothen: For to have been patients, you know, I have to consider more

105 00:17:01.560 --> 00:17:07.410
Alex Pothen: So in this example. This is an authentic path of length three, because they can be solved by swapping matching and non matching edges.

106 00:17:07.650 --> 00:17:14.820
Alex Pothen: I can increase the the cardinality of their matching and therefore, the way to the matching. In this case, and here there are most to non matching edges.

107 00:17:15.810 --> 00:17:22.590
Alex Pothen: And then here's a way to increasing path with Atmos to non matching it just again by swapping matching and non matching edges quite could increase the way

108 00:17:23.280 --> 00:17:29.250
Alex Pothen: That this turns out that these forecasts are all that I need to consider to for to have mutations.

109 00:17:29.880 --> 00:17:36.840
Alex Pothen: But of course you could think about doing this well killed mutations. And so we're at most kids are at most keen on matching edges.

110 00:17:37.380 --> 00:17:47.550
Alex Pothen: Okay, so here's a theorem says if a matching does not admitted compensation, then in India it is ok ok plus one approximation to maximum what a traitor matching. Okay.

111 00:17:48.120 --> 00:17:55.830
Alex Pothen: So if you put k equal to one, then you can have approximation with them so simply find a you know a maximal set of

112 00:17:56.190 --> 00:18:01.680
Alex Pothen: Edges but chosen and careful way because we are choosing to match from the heaviest put vertices

113 00:18:02.190 --> 00:18:07.410
Alex Pothen: Down to the you know the latest, greatest season so on and if you just take one to two, then you get two thirds approximation.

114 00:18:08.130 --> 00:18:14.130
Alex Pothen: So all you really have to do is to just keep looking for medications if there isn't one, then you're done, you have to start the process.

115 00:18:14.970 --> 00:18:25.260
Alex Pothen: So now you're looking for short augmenting paths and short wake increasing paths. And so this is a chance that you can do this in parallel, just to implement this in title. That's what we do.

116 00:18:25.830 --> 00:18:31.860
Alex Pothen: Okay, now turns out that's such a theorem is true for maximum cardinality matching and maximum edge weighted matching as well.

117 00:18:32.130 --> 00:18:39.840
Alex Pothen: But you have to define killed mutations in an appropriate way that's, you know, the set of guided meditations becomes, you know,

118 00:18:40.140 --> 00:18:48.420

Alex Pothen: For X rated matching it actually becomes a little richer, that there are other things that you need to consider, for example, you have to consider some cycles and things like that. But I don't need to do that here.

119 00:18:49.800 --> 00:18:58.560
Alex Pothen: Okay, so, so then we have an algorithm. So here's a key over k plus one approximation algorithm for matching works with matching the serial algorithm. So

120 00:18:59.130 --> 00:19:11.610
Alex Pothen: Typically you initialize with an approximation Matt Clark County match him because I cannot imagine turns out to be easier to do so on these sort of ideas will work the ideas that I just don't talk to you about

121 00:19:12.300 --> 00:19:19.470
Alex Pothen: You know will work in that case as well so you can initialize so that's important practically to get really fast and times.

122 00:19:19.980 --> 00:19:31.140
Alex Pothen: And then why lucky augmentation exists search for experimentation P from an unmatched vertex you and then if he has found, go ahead and open the current matching and update the set of one match. Mrs.

123 00:19:31.620 --> 00:19:37.920
Alex Pothen: Notice that I don't say anything about how you should choose you. I mean, you can choose any on match what next. So this is really what

124 00:19:38.340 --> 00:19:45.420
Alex Pothen: Makes it possible for this algorithm to be parallel and and this is what john with my student. I'm a doubt hurts. That'd be published this year.

125 00:19:46.230 --> 00:19:55.200
Alex Pothen: And then here. So, you know, so for small k, we can hope to implement this in parallel. Right. And so here's a equal to two thirds approximation algorithm internal

126 00:19:55.800 --> 00:20:03.990
Alex Pothen: I'm not giving you a lot of details here, but I'm just sort of giving you a high level view of the algorithm. So again, initialized with an approximate cardinality matching in computer again panel.

127 00:20:04.530 --> 00:20:08.310
Alex Pothen: And if I do on petition exists. So this is, I'm thinking about chairman ray machines.

128 00:20:08.940 --> 00:20:17.820

Alex Pothen: in peril do search for a termination fee from an unnatural tax if p is found. Now we have to lock the vertices, because we have to make sure that

129 00:20:18.630 --> 00:20:25.950
Alex Pothen: You know, different threads. Don't try to augment along different paths and then they run into overlapping

130 00:20:26.460 --> 00:20:36.690
Alex Pothen: PubMed incarcerate increasing guts. So, but it's a lobster obtain, you can go ahead and augment and then you know once you are ready to release our locks you update the setup on match vertices

131 00:20:37.200 --> 00:20:42.060
Alex Pothen: So if you have use of mending pads, then you never unmatchable attacks.

132 00:20:42.870 --> 00:20:54.480
Alex Pothen: But if you use weight increasing that you do as I showed you. And therefore, you may have to update the set of on match where it is and so on. And then, you know, keep looking from unmatched words export to other magicians and then that's

133 00:20:56.010 --> 00:21:01.320
Alex Pothen: Okay, now here's the interesting part. So if the algorithm that one needs to pay attention to.

134 00:21:02.160 --> 00:21:17.070
Alex Pothen: So we can in fact create LifeLock on deadlock and starvation in this context. If you are countless about the way you go about doing your own limitations. Right. So I have an example here where I have created a cyclic graph.

135 00:21:18.150 --> 00:21:29.040
Alex Pothen: That can cause a cyclic. Wait, so here it is. I have you one V1, V2, V3, V4. So this is a weight increasing path of length for we're increasing path of length for

136 00:21:29.940 --> 00:21:33.870
Alex Pothen: Here's another week, increasing path of length for that sort of overlaps with the

137 00:21:34.350 --> 00:21:41.880
Alex Pothen: Previous one and so on. So I have these great increasing paths that overlap. And then finally, this one here that overlaps now way.

138 00:21:42.270 --> 00:21:49.620
Alex Pothen: Right. And now I can create a symbolic link if I'm careless about the way I go about walking through the seas, you know, on, on painting paths.

139 00:21:50.010 --> 00:21:57.330
Alex Pothen: So this picture actually shows you that. So here's the thread to one that is trying to augment along increase the weight along this path.

140 00:21:57.690 --> 00:22:09.060
Alex Pothen: And it goes ahead and locks. This works. But before I can walk this works, you know, another thread to to, you know, augments along this path and locks that projects and so on. So you see the problem, you know,

141 00:22:09.480 --> 00:22:16.980
Alex Pothen: So they have all locked one vertex. Nobody can make progress. They all unlock their go back and locking again. So this is called

142 00:22:17.940 --> 00:22:27.900
Alex Pothen: Know, but there are plenty of documentation to do but nobody can make any progress because of the SEC liquid okay now turns out that you can create cyclic wait for augmenting path of length three as well.

143 00:22:30.420 --> 00:22:32.700
Alex Pothen: And so there is an appropriate locking protocol.

144 00:22:33.390 --> 00:22:42.510
Alex Pothen: I won't go to the details here, but all you have to do is to kind of make sure that you are careful about the way you lock things. For example, if you have an augmenting path. We don't want

145 00:22:43.200 --> 00:22:51.240
Alex Pothen: Vertices at either end, trying to augment along the same path. So we let one of them high priority. So the more number one gets to augment the other one doesn't.

146 00:22:51.690 --> 00:23:00.990
Alex Pothen: And if you have to lock a match edge. You know, we lock the lower number than point. So we look at the ordering of V2, V3 covert slower gets to lock that edge and so on.

147 00:23:01.920 --> 00:23:11.310
Alex Pothen: And and in this, in such a case what we would do is we would first lock be one, then we would lock before the nude lock the minimum of 323 and that that lock the path for us.

148 00:23:12.180 --> 00:23:22.350
Alex Pothen: If that's a augmenting part. Here's a way to increase in path. So here, you know, we have to match edges. And so we look at the n minus a minimum of V1 and V2.

149 00:23:22.710 --> 00:23:28.650
Alex Pothen: An empty is a minimum of weekly and before it's just a lower number 10 points of a match matching adds that we will lock.

150 00:23:29.130 --> 00:23:37.890
Alex Pothen: And now we'll find a minimum of M one M two and the maximum of m&m to and then walk again in order you and then these vertices

151 00:23:38.790 --> 00:23:48.660
Alex Pothen: Now if you do that, you can prove that this locking protocol will avoid deadlock like locking starvation. So here's the theorem that says if there is it to have condition in the graph.

152 00:23:49.260 --> 00:23:56.340
Alex Pothen: In every iteration of the parallel at least one thread from every part of situation cyclic wait graph will succeed in matching

153 00:23:57.480 --> 00:24:10.890
Alex Pothen: So the algorithm that I showed you in in every iteration, at least you know at least one thread. But in fact, you don't, if there's a plenty of order to join sick. Look. Wait, then you know one from every one of those

154 00:24:11.580 --> 00:24:23.670
Alex Pothen: cyclic graph will succeed and therefore the algorithm will keep making progress. So if you want just scan spin forever. So we don't have any of these problems with the appropriate locking protocols.

155 00:24:24.720 --> 00:24:30.000
Alex Pothen: Okay, so let me show you some cereal results. And I've already shown you in the very first slide.

156 00:24:30.240 --> 00:24:36.480
Alex Pothen: To motivate the problem, you know that there are problems graphs on which you can the exact algorithm to take hundreds of hours so

157 00:24:36.720 --> 00:24:43.860
Alex Pothen: Here I'm choosing problems with 100 million vertices and about a billion edges where the exact algorithm actually terminates.

158 00:24:44.220 --> 00:24:50.430
Alex Pothen: So, and then I'm taking the geometric mean of the results on 12 graphs and I'm. That's what I'm showing you

159 00:24:51.150 --> 00:24:57.300
Alex Pothen: And I'm comparing three different algorithms for edge weighted matching with three different algorithms of vertex period matching

160 00:24:57.750 --> 00:25:10.380
Alex Pothen: Now I didn't tell you this but you know clearly if you have a virtuous where the matching problem. You can take the weights of the vertices, add them and assign them to the edge. So take the end point of it. So here's an SUV.

161 00:25:12.720 --> 00:25:20.190
Alex Pothen: Right. Add the weight of Wu plus WV and assign that to the edge so you know made an edge weighted matching problem.

162 00:25:20.820 --> 00:25:29.370
Alex Pothen: It turns out this is a good way to create really hard just problems fridge rater matching problems because we put random weight on the vertices. And you do this for

163 00:25:30.060 --> 00:25:36.420
Alex Pothen: You you create such a problem, you know, these algorithms will take forever to terminate their data matching problems would take forever.

164 00:25:38.100 --> 00:25:45.870
Alex Pothen: But, but anyway. So here are the edge weighted matching algorithms approximation algorithms. And here are the vertex period matching approximation algorithms.

165 00:25:46.230 --> 00:25:56.340
Alex Pothen: So this is the one minus episode approximation algorithm you to do an entity that I mentioned a little bit earlier. And then here you know the other algorithms. So this is a half approximation now.

166 00:25:57.750 --> 00:26:12.000
Alex Pothen: And then here are the vertex weighted matching algorithms. And so this is the the based on the Metroid idea. So the two towards that only uses augmenting paths and then these are the orientation desktop based algorithms that I just asked you about

167 00:26:13.170 --> 00:26:21.780
Alex Pothen: These algorithms that you can see are really, really close to optimal so so one way to measure how good they are used to measure this gap to a community, which is

168 00:26:22.470 --> 00:26:32.100

Alex Pothen: One minus the weight of the approximate matching divide by the way to the maximum matching. So, you know, so if this is like 1% like point 01 and then multiply by 100 so this is that you get 1% or so.

169 00:26:32.790 --> 00:26:35.460
Alex Pothen: And you can notice that if you look at the gap to optimize

170 00:26:36.150 --> 00:26:46.080
Alex Pothen: The two thirds of condition algorithm that be designed has about, you know, point 08 percent or so. So that's really, really close to optimal but they're all doing fairly, fairly well actually

171 00:26:47.070 --> 00:26:57.540
Alex Pothen: And. And if you look at the run times the one minus approximation algorithm essentially slower than the exact algorithm. So it's Georgia so scaling best for them it's a fairly sophisticated algorithm.

172 00:26:58.170 --> 00:27:07.590
Alex Pothen: Implemented cheerily here and the two thirds algorithm is just marginally faster but you know the other algorithms that happen transmission algorithms and the two thirds transmission algorithms.

173 00:27:08.220 --> 00:27:14.970
Alex Pothen: Are you know about 2200 times faster than, than the exact exact opposite put this into problems.

174 00:27:15.360 --> 00:27:26.460
Alex Pothen: Of course, these numbers will vary a little bit depending on which set of problems you choose, as I said, I showed you a problem where in fact the exact number terminates so that desire to performance. He could be infinite. You know those problems.

175 00:27:27.900 --> 00:27:36.930
Alex Pothen: Okay, so that was the first problem that I wanted to talk about. And now I want to talk a little bit about the edge weighted matching problem and to be matching prop.

176 00:27:37.920 --> 00:27:48.840
Alex Pothen: But before I plug and chug right straight ahead. I'll take a second to give everybody a chance to brief and also to ask if there any questions.

177 00:27:52.950 --> 00:27:58.050
Julian Shun: If anyone has any questions, feel free to speak up or you can also type in the chat.

178 00:28:02.790 --> 00:28:07.950

Julian Shun: There's a question from Utah long, how many cores are used in the experiments.

179 00:28:08.850 --> 00:28:17.760
Alex Pothen: Yeah, so I didn't show you the results there. But for example, I'm going to talk about the be matching problem and you will see that I use. I like 16,000 cores, so I will get to them.

180 00:28:19.110 --> 00:28:19.590
Julian Shun: So they're

181 00:28:19.650 --> 00:28:22.800
Alex Pothen: Not all of these, not all of these problems actually have

182 00:28:23.460 --> 00:28:32.550
Alex Pothen: So the results that I showed you for the vertex right. Imagine problem. These are not just a serial results. It's not our results. We do have power results on chairman MRI machines with about 20 minutes or so.

183 00:28:33.480 --> 00:28:44.910
Alex Pothen: But in fact, there's no reason why we can we can implement them, you know, with something a little more effort on larger numbers of course and for the be makin problem. In fact, I'll show you that we have 16,000 courts and I'll show you.

184 00:28:47.550 --> 00:28:47.760
Julian Shun: So,

185 00:28:47.820 --> 00:28:55.080
Alex Pothen: Thank you. Thanks. Okay. Alright, so I'm going to continue. And so here, so we have

186
00:28:55.770 --> 00:29:03.510
Alex Pothen: Integrated maximum edge way to be matching problem. So notice me now have a function be and that could depend on the particular vertex v.

187 00:29:03.840 --> 00:29:13.200
Alex Pothen: And then again we have to say, I must be big. So before we said at most one edge. Now he said most be the edges. And so here unfortunately these colors and not terribly

188 00:29:13.980 --> 00:29:19.020
Alex Pothen: Different. But, you know, so this is my my to matching in this graph, those edges right

189 00:29:19.800 --> 00:29:25.380
Alex Pothen: And notice that this works only has one green agents that are not sort of this one, but I really can't increase the

190 00:29:25.650 --> 00:29:32.310
Alex Pothen: Size of the cardinality that matching because if I just, if I increase this one that I wanted to constrain that there are only two green adjustments on that.

191 00:29:32.910 --> 00:29:42.030
Alex Pothen: On this works. Similarly, if I choose this one I wanted to constrain the, the only two at most two it green or just internet on that vortex, and so on. So, in fact, I can do any better on the screen.

192 00:29:43.170 --> 00:29:47.280
Alex Pothen: Okay, but for a simple example. I made be uniform, but it doesn't have to be.

193 00:29:48.630 --> 00:29:59.490
Alex Pothen: Alright, so again, the first question I would ask is what's the underlying communitarian structure. He said some structure that we can exploit or that they can work off of it. No, and the underlying

194 00:30:00.750 --> 00:30:06.480
Alex Pothen: Structure here. You said to extendable system. Right. So here, this is a relaxation of a

195 00:30:06.900 --> 00:30:12.360
Alex Pothen: Metroid right here you have a set of edges. So before we had the sort of vertices, but now you have a set of edges.

196 00:30:12.660 --> 00:30:24.720
Alex Pothen: And we have substance of edges and again we will say a substance of edges belong to this collection of, you know, sort of us to extend about system in the following way. So I give you tubes will say that, you know,

197 00:30:25.920 --> 00:30:38.670
Alex Pothen: Set belongs to that collection of matching. So, so we just look at subsets of matching engines. That's all we look at. So if there is a matching, then all they're just in their belong to belong to some some some set here.

198 00:30:39.030 --> 00:30:44.790
Alex Pothen: It's a collection of those centers. So I give you a to be matching and one of them to an empty container that one.

199 00:30:45.420 --> 00:31:00.540
Alex Pothen: And there's an edge, he that does not belong to him, too. And so add one plus is a be matching but empty plus ease not then you know Metroid case. All you have to do is to remove one edge and you will be able to make that work. But in the case of

200 00:31:01.590 --> 00:31:09.480
Alex Pothen: To extendable system, you have to remove about two at most two edges and then you know you have a be match.

201 00:31:09.900 --> 00:31:15.690
Alex Pothen: So again, in this case, I can draw you a picture. So here's a vertex you and here's a vertex we

202 00:31:16.020 --> 00:31:26.940
Alex Pothen: And it turns out that. Me too. So this is the edge P M two plus, he is not a matching and be matching. Well, it could be because there's some edge here and there's some edge here.

203 00:31:27.480 --> 00:31:35.250
Alex Pothen: That why so adding this edge violates the be constraints on you and the B complex and be well okay I kind of wanted from here and I'm going to move one edge from there.

204 00:31:35.520 --> 00:31:51.630
Alex Pothen: And then I will be able to satisfy be matching constraint again. And so that's the, that's the result here that I remove at most two edges. And so that makes it to external okay and and so it's quite easy to see that would be matching is it or one matching is also a tournament system.

205 00:31:52.710 --> 00:31:57.630
Alex Pothen: Now, Julian mastery prove the theorem, that's us for a character and have a system.

206 00:31:58.170 --> 00:32:08.640
Alex Pothen: Where you know the size of why now is less than case or vetoed by less than equal to for it to external system but ok external system, the greedy algorithm is the one okay approximation to the optimal solution. Okay.

207 00:32:09.060 --> 00:32:14.970
Alex Pothen: And Tesla greedy algorithm for for for be matching. It's to live case too, so we have a half approximation now.

208 00:32:15.840 --> 00:32:27.540

Alex Pothen: Now this is not the metro duty algorithm. This greedy algorithm is a greedy algorithm that simply chess. Okay, can I add this edge and create satisfy the BU and BV constraint. If so, I've added. If not, I want

209 00:32:27.840 --> 00:32:36.480
Alex Pothen: So it's not looking for amending power to read increasing card. So none of that none of those things at all. It's just, just choosing a maximum set of edges to had to, to the matching

210 00:32:37.500 --> 00:32:46.530
Alex Pothen: Okay, right. So again, the beauty of them. It's simply control just annoying increasing order weights to, again, you have to sort them and control them in that particular order.

211 00:32:46.860 --> 00:32:54.090
Alex Pothen: And then you can add an edge in the matching fits that a spice to be constraints. So know searching for augmenting paths are really interesting parts or any of those things at all.

212 00:32:54.720 --> 00:33:04.440
Alex Pothen: And again, this algorithm, you know, you can implement it in towel. But it turns out that there is a an algorithm that increases the concurrency, much, much more. And that's what I turned to next.

213 00:33:04.950 --> 00:33:11.970
Alex Pothen: And that algorithm is called a pseudo algorithm. And this is due to a colleague credit minor at the University of Bergen, Norway,

214 00:33:12.450 --> 00:33:17.040
Alex Pothen: And mounted on a panel or former PhD student who is now at Pacific Northwest National

215 00:33:17.910 --> 00:33:25.110
Alex Pothen: And this algorithm is very similar to the stable matching algorithms that you may have heard of, definitely. It's very famous because um

216 00:33:25.500 --> 00:33:41.220
Alex Pothen: Let's see. Gail unfortunately passed away, but Shapley won the Nobel Memorial Prize in Economics in 2012, I believe, for this work that they did in the 60s, right. So this is similar to the stable matching algorithms of Galen Shapley and look at the end goal.

217 00:33:42.270 --> 00:33:48.960
Alex Pothen: And so it's based on making proposals and the whole idea is that anybody can make proposals so

218 00:33:49.320 --> 00:34:02.310
Alex Pothen: You just have to make sure that you proposed in a particular order to your neighbors. So proposed your neighbors and non increasing order. The weights. So essentially you have a ranking as you would have in a stable matching problem because us a way to the ranking and then you

219 00:34:03.600 --> 00:34:13.080
Alex Pothen: make proposals in in non increasing order of your writings. OK, now the proposals can be accepted or they can be a noun right and what

220 00:34:13.680 --> 00:34:23.100
Alex Pothen: What happens is that each works keeps track of the best proposal, it has received. Now I'll draw a picture on the next slide. So just bear with me for a second here while we go to this algorithm.

221 00:34:23.550 --> 00:34:31.620
Alex Pothen: So a vertex you propose this to vertex v. If it can be the best offer the economy has number that everybody keeps track of the best offer that they have

222 00:34:32.100 --> 00:34:40.680
Alex Pothen: And if we can propose and you can be the best offer that we have, then it goes out and announce any previous offer that the house. So we are willing. Now to make proposals.

223 00:34:40.980 --> 00:34:47.100
Alex Pothen: And undo proposals. I know. So we are willing to do work and then say okay this work isn't going anywhere. I will take that back.

224 00:34:47.460 --> 00:34:56.160
Alex Pothen: You know, so on. So, this is this is a big difference between, say, for example, or pending pathway to increasing path algorithm for someone that you're not willing to undo things

225 00:34:56.580 --> 00:35:02.820
Alex Pothen: In general, definitely for building class you're not willing to undo things right human and do a matching edge and make a non matching edge.

226 00:35:03.120 --> 00:35:08.730
Alex Pothen: But in the worst way to match in case, for example, will always make it, make it that the very simple always to match. Okay.

227 00:35:09.300 --> 00:35:18.090

Alex Pothen: All right. And when there's an edge in the matching well when when it, when the endpoints of an of an edge proposed to each other. Good. That then it's a matching right

228 00:35:18.570 --> 00:35:23.460
Alex Pothen: And it turns out that you can show the expected number of proposals, it's not the largest order and law again.

229 00:35:23.820 --> 00:35:35.040
Alex Pothen: For complete by paragraphs. When the edge rates are chosen uniformly at random. Of course, you can choose it waits in such a way that the number of proposals this order n squared. You can you can make it back. Okay.

230 00:35:35.610 --> 00:35:38.130
Alex Pothen: But thankfully doesn't happen very often. Right, so

231 00:35:38.970 --> 00:35:46.590
Alex Pothen: And the other thing you can prove is that the the sort of algorithm which is based on making proposals in fact computer same matching

232 00:35:46.860 --> 00:35:52.080
Alex Pothen: That the greedy algorithm funds, right. So I'm saying it gives you the same approximation ratio of a half

233 00:35:52.560 --> 00:36:00.630
Alex Pothen: But more than that, I'm saying. In fact, if you wait if you break ties in the same order in both algorithms, you'll give you exactly the matching that the greedy algorithm will find

234 00:36:00.990 --> 00:36:07.920
Alex Pothen: So you lose nothing by by doing going to this proposal based scheme, other than the fact that we may have to undo some work.

235 00:36:09.420 --> 00:36:17.100
Alex Pothen: Okay, so here's a picture just to make all of that a little, little clearer here you know we three has already proposed to be to

236 00:36:17.640 --> 00:36:27.240
Alex Pothen: With its way to. And so that's a way to can offer. So the best offer that we to have this now to and we want looks at its neighbors and says, okay, I can offer three to be too.

237 00:36:27.780 --> 00:36:33.660
Alex Pothen: And I can offer want to be you. And therefore, I have to offer you know to be to first and then it looks at this week that

238 00:36:34.110 --> 00:36:45.690
Alex Pothen: We three has already offered and just, I can beat that. So it goes ahead and remove that. And so we three now has to go find another proposal and then we want goes ahead and proposes to to be to right

239 00:36:46.560 --> 00:36:53.820
Alex Pothen: Here's a different situation where we three has weighed three and so it already has offered that wait to to be to

240 00:36:54.660 --> 00:37:03.630
Alex Pothen: And then we went goes themselves. Okay, what's the best I can offer be too. But that's too well. Okay, that's not going to be that. So there's no point in making the proposal, our projected

241 00:37:03.960 --> 00:37:18.360
Alex Pothen: I might as well go ahead and propose to you know the next the next one in my, in my ranked list of neighbors. Right. So, so that's a very simple picture of the algorithm, you know, just to show you what happens in this algorithm.

242 00:37:19.410 --> 00:37:27.180
Alex Pothen: It turns out that this can be implemented in Palo clearly because here you know any vortex can make proposals, they all they have to do is to just make sure that they

243 00:37:27.690 --> 00:37:33.780
Alex Pothen: May proposals and rank order the neighbors. That's really all they have to do, but they can go ahead and propose as much as you want.

244 00:37:34.140 --> 00:37:43.770
Alex Pothen: And so here is some some results on the, on the, for the for the beach or the algorithm for the be matching case and somebody had asked me about how many cores with 16,000

245 00:37:44.250 --> 00:37:50.400
Alex Pothen: Cores on this machine. This is the graph would say you know hundred million edges and a billion.

246 00:37:51.150 --> 00:37:56.220
Alex Pothen: You know, one or 2 billion edges and 100 million vertices and a few hundred million vertices and

247 00:37:56.580 --> 00:38:12.150
Alex Pothen: You know wanted 2 billion edges and and this is a strong scaling. So I'm not making the problem any larger, you know, I'm just solving this problem. Same problem with

from 256 to 16,000 times course and these are sort of synthetic problems which I can make arbitrarily large for this.

248 00:38:13.440 --> 00:38:21.090
Alex Pothen: To the sort of chess. And you can see that I'm getting fairly good speed up strong, strong scale. And so, you know, it turns out that

249 00:38:21.750 --> 00:38:27.270
Alex Pothen: This can be implemented officially in power. Now there's quite a bit of work here in terms of implementing it efficiently in parallel.

250 00:38:27.780 --> 00:38:36.300
Alex Pothen: But, you know, I'm not going to go into that. Because clearly, I don't have time already almost out of time already. Let me quickly see a couple more things and then I'll conclude

251 00:38:37.200 --> 00:38:43.380
Alex Pothen: So it turns out that you can extend what we have done to some modular metrics. So in a sub modular function.

252 00:38:43.920 --> 00:38:52.200
Alex Pothen: You're not now something the weeds but instead. For example, you might be taking the square root. So the weights or something like that. Right. And so this is kind of like a discrete derivative

253 00:38:52.620 --> 00:39:07.140
Alex Pothen: offset by one along and element. He and so this dispute derivative is larger than this script derivative for a set it to for the same element. He where I do is bigger than one. So this is kind of like the law of diminishing returns. Right.

254 00:39:07.950 --> 00:39:15.930
Alex Pothen: And so in the sub modular function. It turns out that the I said greedy algorithm is actually half approximate for maximizing some modular voters were to be managing

255 00:39:16.740 --> 00:39:27.420
Alex Pothen: And there's a nice application. So this is what we are doing currently so nice application that looks at quantum chemistry competitions and power machines where you're trying to load balancer competitions that are needed. Okay.

256 00:39:28.020 --> 00:39:40.890
Alex Pothen: And if you have some modular edge way to be matching, then there is no Metroid, but there is actually an intersection of two matrix. And therefore, you get a one third approximation algorithm and it helps you compete, what I call diverse matching. So,

257 00:39:43.410 --> 00:39:53.580
Alex Pothen: Okay, here's another problem we've actually done some work we have worked on minimum wage VHF our problems. So you can think of these as the complement of a beat matching

258 00:39:53.940 --> 00:40:08.040
Alex Pothen: So here we have again BV is given to us for each word tax and we have the constraint that at least be the edges answer, not on V. So it's not at most BV as it wasn't be matching, but it's at least BV

259 00:40:09.120 --> 00:40:16.050
Alex Pothen: As Andy said color problem. In fact, you know, these green edges actually go create what is called a one edge cover in this in this trauma.

260 00:40:16.680 --> 00:40:31.500
Alex Pothen: And it turns out that this is a problem that's very rich and academic paradox, a solution. So there's a greedy algorithm. Their primal dual approximation algorithms that reductions to matching even the well known nearest neighbor graph construction and factors to approximation for ok

261 00:40:32.610 --> 00:40:37.050
Alex Pothen: We have designed several algorithms here and putting primordial evidence has approximate memberships, we have

262 00:40:37.590 --> 00:40:44.370
Alex Pothen: And we have implemented them on shared memory machines for getting scalable parallel algorithms of disparate memory machines.

263 00:40:44.940 --> 00:40:52.710
Alex Pothen: We have actually done one of these productions to matching. And that gives us a to approximation. In fact, we have used to be sued algorithm that I just talked to you about

264 00:40:53.310 --> 00:40:59.640
Alex Pothen: For this for this problem and we have also use it to solve a problem and data privacy called adaptive anonymity problem.

265 00:41:00.990 --> 00:41:03.960
Alex Pothen: Okay, so let me let me conclude my time is getting

266 00:41:05.130 --> 00:41:12.510
Alex Pothen: Near the end at the end. So we've designed several approximation algorithms will what we call degree constraints of problems.

267 00:41:12.990 --> 00:41:17.940
Alex Pothen: We have seen several different design paradigm to extend the greedy our paradigm using augmentation based

268 00:41:18.390 --> 00:41:28.050
Alex Pothen: On algorithms. We've seen proposal based algorithms. We've seen the primal duel with them and so on. When I said we have seen that at least mentioned, I might have discussed a few of them in detail.

269 00:41:29.250 --> 00:41:41.550
Alex Pothen: And it turns out that algorithms with constant worst case approximate ratios yield nearly optimal solutions fast because he's done near linear time complexity algorithms on both cereal and politicians.

270 00:41:42.240 --> 00:41:45.690
Alex Pothen: And some of these outlets, not all, but some of them are scalable.

271 00:41:46.110 --> 00:41:51.870
Alex Pothen: And can be implemented on this pyramid many machines with 10 cake or some more intact, you know, we'd love to get more course we can.

272 00:41:52.170 --> 00:41:59.010
Alex Pothen: Run them on even larger numbers of course it's not easy to get these large, large number of course on the distribute from the do a machine. OK.

273 00:41:59.580 --> 00:42:09.060
Alex Pothen: So we also looked at the number of applications. So this is a problem that we have not looked at, but a lot of people have. So this is one of the reasons why Botox waited matching is this of interest.

274 00:42:09.360 --> 00:42:17.640
Alex Pothen: Because it arises in Internet advertising problem. And in fact, a lot of work has been done on this problem variant of MBM for

275 00:42:18.270 --> 00:42:25.260
Alex Pothen: For the internet advertising problem. It's called the AdWords problem. And people have looked at online algorithms and screaming at someone

276 00:42:26.130 --> 00:42:37.380
Alex Pothen: For adaptive anonymity. We have actually sold an instance with nearly three quarter million individuals and 500 features in a few minutes are using about 10,000 cores on a on a power machine.

277 00:42:38.250 --> 00:42:54.870
Alex Pothen: We're also working on using this for graph construction of class classification for semi supervised classification problems we have used be matching for network alignment problems. And then I talked to you about using sub modular be matching for load balancing problems.

278 00:42:56.400 --> 00:43:05.880
Alex Pothen: Okay, so, you know, clearly, my goal is to try to get you to read a little bit more. If you're interested in knowing more about these these topics.

279 00:43:06.300 --> 00:43:15.720
Alex Pothen: We have written a survey article on this on that appeared in afternoon America in 2019 so it's a 90 page paper. We don't just survey our work with survey. In fact,

280 00:43:16.470 --> 00:43:23.670
Alex Pothen: The whole area. So in terms of matching problems a cardinal the matching different kinds of weighted matching and so on an edge cover problems.

281 00:43:24.000 --> 00:43:35.190
Alex Pothen: And, you know, so we look at in fact problems, some comments or scientific computing. That's a 90 page article that that service to the field data. It also points you to software and also points you to two

282 00:43:36.030 --> 00:43:42.270
Alex Pothen: Papers and and software that V and other other groups have done so here again. I mean, I won't go through these papers, but you know

283 00:43:42.690 --> 00:43:50.910
Alex Pothen: This this slides will be available to those of you who want to look at it more. So our point you that there are papers, but also there is a software library that you could use.

284 00:43:51.600 --> 00:43:58.320
Alex Pothen: Number of people that we have worked with on a number of problems on this. I just want to highlight two people so amor that hurts is the one who

285 00:43:58.710 --> 00:44:07.260
Alex Pothen: PhD student who finished a year ago. He looked on vertex really matching problems and he is now a professor of computer science at

286 00:44:07.890 --> 00:44:19.500

Alex Pothen: KING five University in Saudi Arabia is from or you've worked on be matching problems and productive anonymity problems and he is now a staff scientists at Pacific Northwest National Lab.

287 00:44:20.160 --> 00:44:28.020
Alex Pothen: And for those is a current student working on his Ph. D. Expected to finish in a year or two, and he's working on BH cover and some modular matching problems.

288 00:44:28.770 --> 00:44:36.180
Alex Pothen: Okay, let me finish by saying that, you know, Julian had said something about this activity groups. I'm activity group.

289 00:44:37.050 --> 00:44:49.980
Alex Pothen: So for those of you who are interested in implementing and applying discrete algorithms to particular problems of interest, you know, this is a great community to belong to

290 00:44:50.640 --> 00:44:55.080
Alex Pothen: There's a conference, the inaugural conference is going to be held in July 2021

291 00:44:56.040 --> 00:45:06.450
Alex Pothen: With sign annual meeting. And if you go to that website you can get more information about it like it's sort of a hybrid kind of meeting with both refereed proceedings and also

292 00:45:07.170 --> 00:45:13.800
Alex Pothen: Talks selected from excellent abstract so you can submit a paper and the paper will be reviewed and then they'll be published proceedings.

293 00:45:14.340 --> 00:45:21.660
Alex Pothen: And then, but also, you know, the sign model is essentially to have more talks with excellent abstracts. So it's a hybrid model that will combines combines both

294 00:45:22.470 --> 00:45:30.600
Alex Pothen: And it turns out my term is almost up. So in fact, by the end of this year, I'll be there's a lot going on. But again, if you want to know more about it, please go to

295 00:45:31.080 --> 00:45:38.400
Alex Pothen: You know, just, just go to the site or go to the same site and look at activity groups. For those of you who are particularly students. I encourage you to join.

296 00:45:39.060 --> 00:45:47.370
Alex Pothen: Your membership is free. Essentially, for joining to activity groups. So, but this might be a place where you might meet

297 00:45:48.060 --> 00:45:55.080
Alex Pothen: Fellow Spirit. Spirit kindred spirits who are interested in the kind of problems that you're looking at. We look at different communities.

298 00:45:55.320 --> 00:46:05.400
Alex Pothen: Coming to a scientific computing power algorithms apply district. Now I will have engineering and so on so many, many of these. So this would be a great place for you to publish you

299 00:46:06.300 --> 00:46:13.020
Alex Pothen: So good. Thank you. I will stop here and take any questions. Thank you. Julian and thank you all for your patience.

300 00:46:13.410 --> 00:46:24.120
Julian Shun: Great, thanks a lot. Thanks for the very interesting talk. So we have time for questions if anyone has any questions, feel free to speak up.

301 00:46:25.320 --> 00:46:28.410
Julian Shun: Or you can type on the chat and I can ask for you.

302 00:46:31.980 --> 00:46:34.380
Julian Shun: Yes, I'll start off with a question that I had

303 00:46:35.400 --> 00:46:51.540
Julian Shun: And this is about the first part of the talk, you mentioned that you could get a k over k plus one approximation where K is the length of the path you were considering I'm wondering if you tried to use of k greater than two.

304 00:46:52.830 --> 00:46:57.090
Alex Pothen: We didn't. And the reason Julian. Is this you, you saw the results that we were like,

305 00:46:57.180 --> 00:47:00.180
Alex Pothen: 99.99% of optimal

306 00:47:00.360 --> 00:47:09.780
Alex Pothen: Yeah, so, so it's not clear what you're going to gain by doing it, number one. Number two, it makes the algorithms more complicated. You saw that I had this even with two thirds, I had this problems with

307 00:47:11.040 --> 00:47:16.530
Alex Pothen: You know, causing deadlock and so on. So I really had to be very careful in that so so as I said, you know, you can

308 00:47:16.890 --> 00:47:20.850
Alex Pothen: You know, you can make the algorithms more complex more complicated and sophisticated

309 00:47:21.270 --> 00:47:29.460
Alex Pothen: It just becomes much harder to implement and it's not clear what you gain in practice. I mean, you know, if, in fact, you know, I'm not saying that this works for all problems, right, I mean, so

310 00:47:29.700 --> 00:47:34.920
Alex Pothen: I'm sure that there could be problems that you can cook up where the approximation gracious are much worse and so on and can

311 00:47:35.640 --> 00:47:44.190
Alex Pothen: And then in that case, it's definitely worth pursuing. But yeah, so. So the same thing with edge weighted matching. There's this one minus epsilon across mission algorithm.

312 00:47:44.760 --> 00:47:54.360
Alex Pothen: Do to set pity and ran one and it uses sort of scaling to compute compute the matching and it has it all the complexity of

313 00:47:55.140 --> 00:48:00.060
Alex Pothen: An exact matching algorithm, you still have to deal with blossoms and you have to process them you have to

314 00:48:00.390 --> 00:48:06.090
Alex Pothen: dissolve them, you know, you have to update your weights and all of those sort of things. And so you notice that when we implemented it.

315 00:48:06.990 --> 00:48:15.570
Alex Pothen: You know, it was actually slower than the exact. So, so, yeah. So, one of the nice things about this proclamation algorithm says that they are

316 00:48:16.020 --> 00:48:29.610
Alex Pothen: They are fairly simple to to discuss and also to implement and so this is what we have done. And then, of course, apparently ism, you do need fairly simple algorithms. So the more complicated algorithms, the heartache becomes implement them efficiently in power.

317 00:48:31.470 --> 00:48:32.940
Julian Shun: Great. Thanks. Hello.

318 00:48:34.380 --> 00:48:36.420

Julian Shun: Anyone else have any questions.

319 00:48:43.290 --> 00:48:57.630
Julian Shun: Yeah. So there's a question from William chat in the chat would locks localization strategies for example hardware lock elation in Intel's TS X improve performance or Jesus paralyzing

320 00:49:00.330 --> 00:49:05.640
Alex Pothen: It might be don't know be having tried. We haven't tried it, the students who did this work is now gone.

321 00:49:06.090 --> 00:49:12.810
Alex Pothen: And go you know and so he's still adjusting to his new job to Saudi Arabia. So we really haven't pursued things much further.

322 00:49:13.590 --> 00:49:25.560
Alex Pothen: I'm pretty sure that that you know better synchronization methodologies and mechanisms will definitely improve performance, that's for sure. But we haven't we haven't tried anything beyond

323 00:49:26.580 --> 00:49:31.260
Alex Pothen: You know, just, just a sort of locks available in open MP until

324 00:49:33.330 --> 00:49:34.770
Julian Shun: Great, thanks.

325 00:49:37.680 --> 00:49:52.200
Julian Shun: I was actually wondering like how much theory and sister between different runs of the like to be suitor algorithms that you were describing since I noticed that it is, it might be non deterministic based on like what order the vertex proposes.

326 00:49:54.090 --> 00:49:59.070
Alex Pothen: Yeah, so again for knowledge problems for very large problems.

327 00:50:01.110 --> 00:50:07.230
Alex Pothen: We haven't seen much of a variance for small problems, maybe, you know, it might, it might be because

328 00:50:07.950 --> 00:50:14.010
Alex Pothen: You know, because of the, the total amount of work that you need to do. I mean, so, so these algorithms are not really

329 00:50:14.850 --> 00:50:24.810

Alex Pothen: Bound so much. Bye bye. The competition there they're bound much more by memory access and you know synchronization and things like that so

330 00:50:25.560 --> 00:50:40.260
Alex Pothen: So yeah, we haven't seen too much variation we have tried petitioning the vertices, you know, randomly randomly reordering the vertices and things like that. We haven't we haven't seen much variation

331 00:50:41.310 --> 00:50:46.530
Alex Pothen: But again, it could be because we have large problems, there's a fair amount of work to do and

332 00:50:47.430 --> 00:50:52.080
Alex Pothen: And and then, you know, so I didn't talk to you about, you know what, what we did to make this

333 00:50:52.530 --> 00:50:59.430
Alex Pothen: Algorithm scalable, you know, and so on. So, so, in fact, you know, you have to do. Super steps you have to organize your algorithms into sort of steps.

334 00:50:59.760 --> 00:51:09.870
Alex Pothen: And then you have to choose a careful granularity. How much do you communicate how often do you communicate courses computer. So you have to set that know fairly well and then

335 00:51:10.740 --> 00:51:16.260
Alex Pothen: And then you have to do. We do essentially a synchronous super steps. So effectively you know there's there's a lot of

336 00:51:16.590 --> 00:51:22.110
Alex Pothen: Because it's a piece of the algorithm, you know, the algorithm, you know, it might just end up doing more work. If you are really bad.

337 00:51:22.440 --> 00:51:27.660
Alex Pothen: About how you, how you use in synchrony. So you know that that's the worst thing that can happen to you, but

338 00:51:27.930 --> 00:51:35.040
Alex Pothen: The algorithm is going to be correct, because in fact the title algorithm is going to compute exactly the same matching that the serial algorithm would compete with also

339 00:51:35.580 --> 00:51:45.900

Alex Pothen: And so because there's an underlying invariant that the algorithm has to satisfy right and so on. So we really haven't seen very much variability and Julian, but

340 00:51:46.440 --> 00:51:55.950
Alex Pothen: But again, that's not to say that can happen. It's just the fact that, you know, the we do these very large runs on fairly, you know, smaller sets of problems and

341 00:51:56.310 --> 00:52:02.640
Alex Pothen: You know, we don't get a huge amount of time on these machines and so on. So it could be the fact that, you know, if he if he had more

342 00:52:03.630 --> 00:52:11.640
Alex Pothen: You know, more sort of opportunities to expand maybe might actually see them but unchecked memory machines for sure, not because we have access to a lot of shared memory machines.

343 00:52:12.210 --> 00:52:25.680
Alex Pothen: You know what would say, you know, we've even run them on on IBM machines with several hundred course so like I don't know 700, of course, or 800 course or something. And we really haven't seen very much very much variance here.

344 00:52:27.420 --> 00:52:30.090
Julian Shun: Okay, great. Yeah, thanks. Thanks for letting me know.

345 00:52:34.650 --> 00:52:36.300
Julian Shun: Anyone else have questions.

346 00:52:40.620 --> 00:52:43.890
Alex Pothen: Maybe I'll just stop stop sharing

347 00:52:46.950 --> 00:52:48.930
Alex Pothen: And maybe you can see at least my picture.

348 00:52:49.770 --> 00:52:50.640
Yeah.

349 00:52:51.720 --> 00:52:52.320
Alex Pothen: Yeah, sorry.

350 00:52:53.130 --> 00:52:53.400
Julian Shun: I don't

351 00:52:54.420 --> 00:53:08.610

Julian Shun: Know, yeah. I actually had another question. So I'm wondering if if these algorithms like warfare. It's a piece out for this would work for solving matching problems on Hyper grouse you've ever looked into that.

352 00:53:09.720 --> 00:53:23.760
Alex Pothen: So I haven't looked at hyper graph matching problems, but I have a colleague of some statistics. Actually, that was defended a few weeks ago at the University of Leo and so

353 00:53:25.650 --> 00:53:26.250
Alex Pothen: His name is

354 00:53:27.300 --> 00:53:30.570
Alex Pothen: My colleague name is Bo char and his student is

355 00:53:32.580 --> 00:53:45.630
Alex Pothen: I know God. I saw the youngest managers that that's his name. So, and, and he he has looked at some very simple. So these are essentially uniform

356 00:53:46.050 --> 00:53:59.310
Alex Pothen: Hyper graphs, you know, things like that. So he has looked at some some matching problems in that context. But, but, yeah, I mean, so definitely some of these ideas can be used in those contexts. But of course, this problem. So it'd be hard.

357 00:54:00.330 --> 00:54:05.160
Alex Pothen: Or intractable. And so, you know what, it's not clear what kind of approximation of someone can you

358 00:54:06.450 --> 00:54:08.670
Julian Shun: Yeah. Great. Great. Thanks a lot.

359 00:54:11.460 --> 00:54:11.820
Julian Shun: Okay.

360 00:54:14.220 --> 00:54:35.460
Julian Shun: Yeah. So it seems like no one has a question right now, but we'll, we'll leave the zoom room open. If anyone thinks have any questions or anyone just wants to chat I I know Jeremy wanted the chat sees here. So yeah, well let's let's think Alex again for the very great talk.

361 00:54:36.930 --> 00:54:37.740
Alex Pothen: Thank you very much.