**MIT CSAIL FastCode Seminar: Professor Kathy Yelick – August 3, 2020**

1
00:00:35.130 --> 00:00:40.830
Julian Shun: Good afternoon.  Today. I'm very happy to have Kathy Yelick as our speaker for the fast code seminar.

2
00:00:41.280 --> 00:00:49.950
Julian Shun: Kathy is the Robert S. Pepper distinguished professor of UCS and Associate Dean for research and the division of computing data science.

3
00:00:50.250 --> 00:01:02.190
Julian Shun: at UC Berkeley. And she's also a senior advisor on computing at Lawrence Berkeley National Labs. Her research focuses on high performance computing programming systems parallel algorithms.

4
00:01:02.520 --> 00:01:12.270
Julian Shun: And computational should know mix and she currently leads the Exabiome project, which is on exhale solutions for microbiome analysis.

5
00:01:13.320 --> 00:01:22.920
Julian Shun: Kathy has received many awards for her outstanding contributions. She's a member of the National Academy of Engineering and American Academy of Arts and Sciences

6
00:01:23.280 --> 00:01:36.420
Julian Shun: She's also a fellow of the ACM and the American Association for the Advancement of sciences and she is also a recipient of the ACM IEEE can Kennedy award as well as the ACM W Athena award.

7
00:01:37.320 --> 00:01:49.200
Julian Shun: Kathy received her PhD in EECS from MIT, and she's been a professor at UC Berkeley since 1991, with a joint research appointment at Berkeley Lab since 1996

8
00:01:49.800 --> 00:01:58.320
Julian Shun: And today, Kathy is going to talk about some of her recent research on genomic analysis and learning at scale. So I'll turn it over to Kathy.

9
00:01:59.520 --> 00:02:07.800
Kathy Yelick: Great. Well, thanks very much for having me. And it's great to see a bunch of familiar faces from way back. And more recently, so

10
00:02:08.220 --> 00:02:11.520
Kathy Yelick: I'm going to talk a little mostly about the Exabiome project and

11
00:02:12.240 --> 00:02:21.450
Kathy Yelick: I decided because this well by the name of the seminar at least I thought you might want to hear more about the algorithm. So I'm going to try to talk a little bit more about how some of these problems get

12
00:02:22.110 --> 00:02:30.660
Kathy Yelick: Get paralyzed and what makes them I think interesting from a parallel algorithms standpoint, in addition to a microbiology standpoint, so

13
00:02:32.550 --> 00:02:39.210
Kathy Yelick: My slide. Okay, so I'm going to basically this is the outline of my talk, I'm going to talk a little bit about some of the science that we're

14
00:02:39.690 --> 00:02:47.940
Kathy Yelick: Enabling with some of these high performance algorithms, a little bit about the machines. I think you're mostly familiar with the direction machines are going, but it will help

15
00:02:48.420 --> 00:03:02.430
Kathy Yelick: Motivate some of the challenges that come up in the algorithms. So, broadly speaking microbiome analysis is important for a lot of applications. I think for many people, especially computer scientists who are

16
00:03:03.210 --> 00:03:13.650
Kathy Yelick: When you hear the word biology or genomics, you immediately think of the human genome, and of course the microbiome is very important in understanding human health that gut microbiome skin microbiome and all these others.

17

00:03:14.370 --> 00:03:24.630
Kathy Yelick: But, but that it also comes up a lot, and applications of interest to the Department of Energy, such as an environmental science in bio energy applications. So how do you break down.

18
00:03:25.080 --> 00:03:38.100
Kathy Yelick: switchgrass for biofuels for bio manufacturing. So trying to also use microbes to produce, whether it's drugs or other kinds of materials or chemicals that you're interested, as well as understanding some of the fundamental

19
00:03:38.580 --> 00:03:42.240
Kathy Yelick: Parts of the tree of life that we don't understand very well so

20
00:03:43.560 --> 00:03:55.410
Kathy Yelick: Micro microbial data is growing dramatically. This is just a graph of the number of meta genomes and meta genome, by the way, is a sample taken from a microbiome.

21
00:03:56.370 --> 00:04:00.750
Kathy Yelick: In the, in the wild, so to speak. So whether it's from a human gut or from

22
00:04:01.050 --> 00:04:11.970
Kathy Yelick: Say a sample of soil, it might have hundreds or even thousands of different species inside of it. So it's not a single genome, but a whole collection of genomes and that sample and that sample is then called a meta genome.

23
00:04:13.080 --> 00:04:21.630
Kathy Yelick: That there's a new project led by Lawrence Berkeley National Lab. The National microbiome data collaboratory which is trying to collect a lot of this microbial data.

24
00:04:22.200 --> 00:04:35.310
Kathy Yelick: Together and and then use it to help understand things like function, but just to annotate and then share more broadly. A lot of the microbiome data with different levels of processing on the data.

25
00:04:36.600 --> 00:04:46.110

Kathy Yelick: So just to give you a sense I talked. I mentioned already, you've got things like human or animal gut microbiome. So here's our cow Roman which is

26
00:04:47.190 --> 00:04:52.080
Kathy Yelick: Being used to understand how you can break down grasses for things like biofuels.

27
00:04:52.500 --> 00:05:05.670
Kathy Yelick: That's one of the sort of medium in terms of complexity and acid mine would have what is simpler. And so it was very complicated. This graph is just showing the number of species per, per meta genome. So the

28
00:05:06.780 --> 00:05:15.210
Kathy Yelick: So as an acid mind might only have a few different microbes that live in that particular environment. Whereas when you get to soil, you can have

29
00:05:15.750 --> 00:05:20.160
Kathy Yelick: Thousands of species that are in them would make it, making it much harder to analyze the data.

30
00:05:21.060 --> 00:05:25.680
Kathy Yelick: So the Exabiome project is one of the access scale projects funded by the Department of Energy.

31
00:05:25.950 --> 00:05:38.190
Kathy Yelick: It is the full title is excess scale solutions for microbiome analysis. And the idea is to take access scale algorithms and system to solve problems that were previously intractable. And there are really three

32
00:05:38.610 --> 00:05:45.150
Kathy Yelick: Pillars of this project. The first one and the one I'm going to talk about the most today is meta genome assembly.

33
00:05:45.630 --> 00:05:57.570
Kathy Yelick: So that's the problem of taking raw sequence data and I'll say more about this in a minute and turning it into genomes, or at least larger fractions of genome so that you can do things like find the genes that are in those genomes.

34
00:05:58.710 --> 00:06:04.050
Kathy Yelick: It uses lots of different data structures hash tables, graphs alignment algorithms and as

35
00:06:04.770 --> 00:06:14.160
Kathy Yelick: That I will talk in more detail about I'll mention briefly and i think i can village has been has spoken at this seminar, not too long ago, and I assume he talked more about

36
00:06:14.820 --> 00:06:20.940
Kathy Yelick: Protein clustering and hip MC I'll, I'll mention it a little bit more at the end just because of some of the similarity in some of the

37
00:06:21.480 --> 00:06:30.840
Kathy Yelick: computational problems that come up. And the last one which I won't say too much about is comparative analysis. And this is a part of the project will be led by Los Alamos National Labs.

38
00:06:31.290 --> 00:06:41.280
Kathy Yelick: But this is where you're given say too many genomes and you're trying to figure out what is similar or different across them. You can use that for things like trying to understand after a

39
00:06:42.090 --> 00:06:53.520
Kathy Yelick: After a fire whether there's a different microbes in the soil than before a fire has come through, or many other sort of environmental and monitoring sorts of applications.

40
00:06:54.360 --> 00:07:04.800
Kathy Yelick: So this is our timeline. I won't go through this in detail, except as an excess scale projects. We have to have a pretty clear set of goals as as not just sort of an open ended research project and so

41
00:07:05.250 --> 00:07:12.810
Kathy Yelick: We have goals with respect to both assembly and clustering, as well as comparative analysis.

42
00:07:13.530 --> 00:07:24.630
Kathy Yelick: Which is, and this is giving us a little bit of quantitative idea of the size of things that we were able to assemble and cluster at the beginning. So about a quarter of a terabyte was the largest assembly.

43
00:07:25.170 --> 00:07:33.060
Kathy Yelick: That's because the the assemblers the production assemblers that people have been using. We're all running on shared memory machine. So if you assume that you can get about

44
00:07:33.510 --> 00:07:44.160
Kathy Yelick: Maybe one to two terabyte shared memory machine, the size of a data sets that you can assemble on that because it does blow up in the middle of the computation is about a quarter of a terabyte

45
00:07:44.880 --> 00:07:58.770
Kathy Yelick: The clustering problems, they were able to add the giant Genome Institute, which is at Lawrence Berkeley National Lab, they're able to cluster about 15 million proteins. It took them about 15 weeks of computing time to do that. They, they are

46
00:07:59.880 --> 00:08:08.700
Kathy Yelick: The computational biology and community in general is amazingly patient with waiting for these things to complete and will dedicate enormous amounts of

47
00:08:09.090 --> 00:08:17.490
Kathy Yelick: single node computing time but often had not considered high performance, distributed memory that is sorts of computing. And many of these algorithms.

48
00:08:18.270 --> 00:08:23.520
Kathy Yelick: In some cases, they're they're sort of natively parallel and those also do use many different kinds of clusters, but

49
00:08:23.970 --> 00:08:28.050
Kathy Yelick: In particular, and clocked in this protein clustering animated genome assembly. Neither of them were

50
00:08:28.320 --> 00:08:41.640
Kathy Yelick: Were running and distributed memory before our goal is to be able to assemble a 50 terabyte meta genome and to cluster 50 billion proteins. And we've gotten a certain way is along the way so far. So what do I mean by assembly.

51
00:08:42.780 --> 00:08:56.550
Kathy Yelick: So we've got our sequence or spits out fragments of a genome, even if you're doing single genome assembly, like a human genome. It spits out these fragments and it inserts errors into it. So we've got little inserts in here, we've got

52
00:08:57.780 --> 00:09:01.740
Kathy Yelick: characters that are mismatched we might have deletions and things like that.

53
00:09:02.040 --> 00:09:13.410
Kathy Yelick: And then the goal of assembly is to turn this into ideally a complete genome in practice, we will we will not be able to get complete genomes out, but we'll get long enough fragments that from that we can do things like find

54
00:09:13.680 --> 00:09:26.760
Kathy Yelick: Genes. So one of the tricks is to is to read it multiple times. So a typical number for something like a human genome, would be to to make have 20 copies 20 sort of reads of the genome.

55
00:09:27.210 --> 00:09:36.600
Kathy Yelick: With the idea that you'll have multiple copies, then, of any particular instance of the genome and then be able to decide to resolve any errors that occurred in any one of the reeds.

56
00:09:38.850 --> 00:09:44.730
Kathy Yelick: So this is akin to putting together a puzzle where you don't know what the puzzle cover looks like. So you

57
00:09:44.940 --> 00:09:54.900
Kathy Yelick: Don't have a picture of it that's actually not true in the human genome, you do have the cover. You have you have a reference genome that's been previously assembled, you can take the pieces you can line them up.

58
00:09:55.320 --> 00:10:01.050
Kathy Yelick: Find the place in the puzzle, where they belong, and that speeds up considerably from an algorithmic standpoint, but

59
00:10:01.500 --> 00:10:12.780
Kathy Yelick: Just makes it much simpler from it from a parallelism standpoint as well. But we're looking at the de novo assembly problem where you don't have a reference Gino this comes up in many plant species as well as

60
00:10:13.500 --> 00:10:17.100
Kathy Yelick: Def definitely comes up and all these environmental meta genome samples.

61
00:10:17.700 --> 00:10:24.180
Kathy Yelick: So that, but that problem that I mentioned of assembly is taking these this de novo problem where you don't have a reference

62
00:10:24.600 --> 00:10:36.360
Kathy Yelick: So this is some examples from the plant biology group at jgr led by Dan Rockstar who's also on faculty UC Berkeley and some of the larger data sets that they've assembled using our parallel assembler

63
00:10:36.810 --> 00:10:46.140
Kathy Yelick: Which was based on their assembler of their shared memory assembler called miraculous we built originally version called hipper, which it was used to do this kind of

64
00:10:47.130 --> 00:11:03.480
Kathy Yelick: Assembly these the numbers in this table, give you some idea of the size of the output versus the size of the the estimated size of the genome. And so you can see we're getting about half of the the genome covered by the assemble genome.

65
00:11:05.040 --> 00:11:15.210
Kathy Yelick: And this has been used by groups to do a number of different things. One of them is some work by Sean Gordon, who at the time was at AGI looking at

66

00:11:16.110 --> 00:11:21.300
Kathy Yelick: A PAN genome assembly. So trying to understand for a population of

67
00:11:21.720 --> 00:11:29.340
Kathy Yelick: genomes that are closely related. So within a species, but many different strains of it, looking at how how these different strands divert

68
00:11:29.550 --> 00:11:39.330
Kathy Yelick: The problem is if you if you do what we often do with human genome data which is not to do a de novo assembly, but to align it against the reference, which is, as I said, faster and easier.

69
00:11:39.780 --> 00:11:52.050
Kathy Yelick: The problem is you will often sort of damp it out. The, the variations across the strains and so it's actually useful to do a from scratch de novo assembly on each one of the different

70
00:11:53.490 --> 00:12:00.750
Kathy Yelick: Strains so that you can really get a better idea of what the differences are across the different species of the different instances of the species.

71
00:12:02.340 --> 00:12:05.430
Kathy Yelick: Now the main focus, though, of our project is actually microbes and

72
00:12:05.880 --> 00:12:19.020
Kathy Yelick: They don't occur as a single species in isolation out in the wild. And in fact, many of them you can't grow in a laboratory environment or haven't been growing in the laboratory environment. So now we've got the problem of this puzzle construction.

73
00:12:19.440 --> 00:12:31.170
Kathy Yelick: Where we don't have the cover of it. We also have actually many different so say thousands of different puzzles all mixed together in the case of soil where we've got 1000 different species. So we're trying to reconstruct each one of those puzzles.

74
00:12:31.770 --> 00:12:37.740

Kathy Yelick: And to add to the complexity of this, the species will occur in different abundances some

75
00:12:38.610 --> 00:12:44.610
Kathy Yelick: Times vary widely different abundances within that sample. So you can more easily assembled things that are very

76
00:12:44.910 --> 00:12:53.730
Kathy Yelick: Abundant in this but but trying to get to the rare species which can still be very important and understanding what the functional behavior of that meta genome is

77
00:12:54.150 --> 00:12:58.500
Kathy Yelick: Will be much harder because there just aren't as many copies of those puzzles pieces, if you will.

78
00:12:59.340 --> 00:13:10.020
Kathy Yelick: And so one of the examples was looking at an environmental genome. This is looking at the Twitchell wetlands, which is an area in Northern California that has been that was

79
00:13:10.890 --> 00:13:18.900
Kathy Yelick: Was originally a freshwater environment and then had become salt water, and then they through environmental remediation have

80
00:13:19.290 --> 00:13:28.740
Kathy Yelick: Turned it back into fresh water, but we're then able to compare what do the microbes what they look like in the salt water versus freshwater environment and

81
00:13:29.340 --> 00:13:38.940
Kathy Yelick: And the question. One of the key questions is how much carbon is sequestered or released from those different in those different types of environments. And I think one of the things they found was that it was

82
00:13:40.440 --> 00:13:47.610
Kathy Yelick: Surprisingly, I get at least to me, the, the salt water environment actually absorbed more carbon than the the freshwater one did.

83
00:13:48.720 --> 00:13:55.620
Kathy Yelick: And this was so this was one of the data sets. So that we've looked at assembly from the original science paper, they had done it with another assembler

84
00:13:56.100 --> 00:14:03.300
Kathy Yelick: But we've been looking at this in terms of trying to get in terms of getting higher quality and then faster parallel assemblies out of it.

85
00:14:03.900 --> 00:14:08.460
Kathy Yelick: At this point, our assembler is being used by number of other science groups unrelated.

86
00:14:09.150 --> 00:14:16.440
Kathy Yelick: came to us and and even to LDL at the giant genomes Institute and this just gives you an idea of some of the size of those

87
00:14:16.650 --> 00:14:26.880
Kathy Yelick: Data sets, but also the type of science problems that they're being used for. So this kind of carbon cycling in soil. The third one, or an understanding the wetlands, for example.

88
00:14:27.570 --> 00:14:30.870
Kathy Yelick: Looking at deep decomposition and nitric vacation.

89
00:14:31.380 --> 00:14:39.360
Kathy Yelick: After a fire. So, this is this question of what does it look like, what does the microbiome look like after a fire has gone through a particular area.

90
00:14:39.660 --> 00:14:43.590
Kathy Yelick: And then the last one, looking at biofuels and things like that so

91
00:14:43.950 --> 00:14:51.090
Kathy Yelick: And one of the things that we've found and collaborating with the giant Genome Institute is some of the scientists had been constrained by the

92

00:14:51.420 --> 00:14:55.500
Kathy Yelick: Fact that they couldn't assemble a larger one. So they had been they had not been trying to

93
00:14:55.770 --> 00:15:08.700
Kathy Yelick: Collect such large samples because they didn't think they could actually handle the computational problem. And now that they realize that there's a larger scale assembler available. They have been getting more requests for very large data set sequencing and then

94
00:15:09.390 --> 00:15:20.250
Kathy Yelick: Assembly. So one of the questions that we started with. At the beginning was can you not just handle larger data sets, but can you get more information that is can you do better science by

95
00:15:20.730 --> 00:15:27.540
Kathy Yelick: Having a an assembler that runs across distributed memory and therefore can handle multi terabyte data sets and

96
00:15:28.110 --> 00:15:36.750
Kathy Yelick: We recently published a paper on this is actually not the reference for the latest paper I'm looking at the this quality issue of the graph on the right.

97
00:15:37.020 --> 00:15:49.440
Kathy Yelick: And what we're doing is comparing what people did before. For something like this. This Twitchell wetlands data, which is the data is actually sample multiple times. So I think they started with something like these.

98
00:15:50.790 --> 00:16:03.870
Kathy Yelick: What was this, I guess 15 different assemblies and or 21 lane sorry 21 different assemblies 21 lanes, which are just think of that as sort of something that comes out of the sequence or separately.

99
00:16:04.500 --> 00:16:15.750
Kathy Yelick: A score of goop on the, on the, the, the slide that you're doing the sequencing on and they would assemble each one of those 21 lane separately because that was that was something that would fit within the

100
00:16:16.230 --> 00:16:23.520
Kathy Yelick: The previous assembler, this was meta space. I think was the assembler that they typically use in production for these kinds of problems and

101
00:16:24.000 --> 00:16:29.520
Kathy Yelick: What this graph on the right, with and without going into detail is showing how complete the assembly is

102
00:16:29.820 --> 00:16:40.020
Kathy Yelick: And the number of genomes at a particular completeness. And so the blue line is showing what happens when you co assemble everything together, which you can only do if you have a large scale.

103
00:16:40.470 --> 00:16:56.760
Kathy Yelick: distributed memory assembler versus what happens if you do them one at a time and then if you do them in various, various combinations of things. And so you can get more better quality genomes out of this type of large scale assembly as well.

104
00:16:58.410 --> 00:17:09.300
Kathy Yelick: The second topic that I that, as I said, is in the is in the project is looking at protein clustering, and this will turn into a problem.

105
00:17:10.170 --> 00:17:21.810
Kathy Yelick: You can call it a graph problem. You can call it a sparse matrix problem, but you're starting with a similarity score between protein. So oftentimes these proteins will come from the genes that you find in the same meta genome.

106
00:17:22.200 --> 00:17:27.690
Kathy Yelick: You've got a large database of those you figure out what protein. The protein coding genes code for

107
00:17:28.110 --> 00:17:40.710
Kathy Yelick: Then put those into a similarity matrix and then try to find clusters in that matrix and this part of the project is really led by item bullish and. And as I said, I think he may have have talked about it a little bit more

108
00:17:41.160 --> 00:17:47.880
Kathy Yelick: But this is one of the science results that they got they got out of this working with Nico's creepiness at j CI.

109
00:17:48.690 --> 00:17:58.200
Kathy Yelick: Is looking at how that what what the different clusters of genomes are. And there are other groups that are that are jgr users now Gigi is a user facility.

110
00:17:58.680 --> 00:18:09.630
Kathy Yelick: That are also now using this this type of clustering algorithm. So what are they using this for they try to find new proteins that might have applications such as finding new types of CRISPR genes.

111
00:18:10.170 --> 00:18:20.430
Kathy Yelick: Finding antibacterial antiviral something, of course, we'd be very interested in right now, looking for gene clusters for for new antibiotics and so on.

112
00:18:22.020 --> 00:18:33.270
Kathy Yelick: And then the last part of the project is to compare to meta genomes to each other. There are a few different ways that you can do this. One of them is to sort of figure out who is in the

113
00:18:33.810 --> 00:18:40.500
Kathy Yelick: What what species are in the in the messaging I'm sample, you're looking at in that case you need to have

114
00:18:40.740 --> 00:18:48.870
Kathy Yelick: That species represented in a database. And as I mentioned before, and some of these samples, such as in soil. There's a lot of unknown species. So that has certain limitations.

115
00:18:49.350 --> 00:18:55.530
Kathy Yelick: You're also may look at what they do that is looking at things like the proteins that are coated for in the in the

116
00:18:56.610 --> 00:19:04.170

Kathy Yelick: In the sequences. And in this case, you, you may, it may be useful to do assembly before you try to

117
00:19:04.590 --> 00:19:13.410
Kathy Yelick: Do either the first or the second one of those. The last one actually can be done on raw data, and I'll say more about camera analysis, you can just care comparison of the histograms.

118
00:19:13.710 --> 00:19:26.100
Kathy Yelick: Of all of the fixed length strings for some length k and use that histogram than to try to characterize that meta genome sample and then go back and look at at the differences. This is useful if

119
00:19:27.000 --> 00:19:39.300
Kathy Yelick: If you're it tells you less about kind of why they're different or exactly when they're different, but allows you to then find the differences which you can then go back and do more analysis on the differences and we are building tools for that as well.

120
00:19:40.860 --> 00:19:41.340
Kathy Yelick: So,

121
00:19:42.810 --> 00:19:54.420
Kathy Yelick: And they use them a number of different techniques such as in hashing and then computing something like a joke card distance between the different messaging on this based on these kinds of

122
00:19:55.680 --> 00:19:57.720
Kathy Yelick: The cameras and things like

123
00:19:59.160 --> 00:20:15.900
Kathy Yelick: So a lot of these based on on these different distance metrics. So I'm going to just to give you a sense of some of the scientific problems. I'm happy to. I can't really see all your faces right now because I'm hiding that but I'm happy to take any questions if there are any at the moment.

124
00:20:21.180 --> 00:20:24.570

Julian Shun: Feel free to unmute yourself. If you have any questions.

125
00:20:27.990 --> 00:20:39.360
Kathy Yelick: Okay, well I'm not hearing anything I will continue talking. Let me, I will get kind of quickly talk about the machines because I think you are all familiar with what these machine high performance machines look like today.

126
00:20:39.660 --> 00:20:48.750
Kathy Yelick: And then go on to talk about the algorithms. So just a little bit of history back in 2007 I was involved, to some extent, and

127
00:20:49.500 --> 00:21:00.150
Kathy Yelick: One of the some of the workshops and putting together a report for the access scale program and the title of this interestingly was modeling and simulation for the access scale for the energy and environment. So this was a Department of Energy.

128
00:21:00.450 --> 00:21:10.050
Kathy Yelick: Program. So not surprisingly, a tad energy in the title, but it was entirely focused on modeling and simulation, which had been really the focus of all of the high performance computing work.

129
00:21:10.380 --> 00:21:23.310
Kathy Yelick: In the Department of Energy and the entire Oscar program the advanced scientific computing program at do he has had been really focused almost exclusively on modeling and simulation and modeling and simulation of the largest scales.

130
00:21:23.640 --> 00:21:39.510
Kathy Yelick: And this was, this is just a top 500 graph which you're familiar with, but I would say that HTC and scientific computing were kind of synonymous with simulation, as opposed to being about any other kind of computational problems such as data analysis.

131
00:21:41.370 --> 00:21:47.850
Kathy Yelick: All of you are familiar with, you know, the end of Dennard scaling and of course now the end of Moore's law so

132
00:21:48.480 --> 00:22:01.200

Kathy Yelick: That we're starting to see tailing off already in terms of Transistor density, but clock frequency of course ended in around 2004. So obviously this is leading to a lot of on ship.

133
00:22:01.830 --> 00:22:04.950
Kathy Yelick: Parallelism of various kinds, and on node parallelism and

134
00:22:05.760 --> 00:22:16.170
Kathy Yelick: But just maybe the most interesting thing. This slide is the little comment that I just added, which is although those of us in the community, computer science community are very familiar with this and

135
00:22:16.770 --> 00:22:25.140
Kathy Yelick: I think that even more broadly, not everybody has internalized the end of Moore's law that they are used to the kind of implicitly

136
00:22:25.800 --> 00:22:31.830
Kathy Yelick: The exponential growth that we growth that we've gotten in computing. And one example of this is the Atlas project, one of the

137
00:22:32.100 --> 00:22:37.950
Kathy Yelick: Project science projects that the Large Hadron Collider that realized about a year, year and a half ago that they had

138
00:22:38.340 --> 00:22:46.950
Kathy Yelick: Underestimated their computing costs by about a billion dollars. And that was because they were just running the projections out from history and of course

139
00:22:47.280 --> 00:22:58.920
Kathy Yelick: They were not going to get the same amount of computing for per dollar that they were expecting. And they had not paralyzed. These codes necessarily for distributed memory and certainly not optimized them for things like GPUs.

140
00:22:59.850 --> 00:23:11.250
Kathy Yelick: So at the beginning of the access scale project. There were three swim lanes. One of them was faster clocks and that one was kind of immediately discarded as something that was never really going to be possible.

141
00:23:11.760 --> 00:23:24.120
Kathy Yelick: So we weren't that naive in the beginning discussions of this. The other one was 100 times more cores and maybe 10 times more computing cabinets and the and the last one here was GPUs. And at this point, the

142
00:23:25.140 --> 00:23:31.440
Kathy Yelick: You know that the largest machine at at nurse and I'll show results from the Corey supercomputer, which has nice landing processors.

143
00:23:31.770 --> 00:23:43.080
Kathy Yelick: And that was really the focus of what we had thought we were targeting in the zoo by on projects so many, many cores of 68 and the current system but you know over 100 cores may be purchased

144
00:23:44.250 --> 00:23:53.550
Kathy Yelick: But traditional sorts of shared memory architectures really has not been what you know what it looks like will be getting now that the the the new

145
00:23:54.030 --> 00:24:03.090
Kathy Yelick: Japanese machine is of course based on that model with ARM processors, but we're really looking at accelerators and specifically GPUs for the US access to machines.

146
00:24:04.320 --> 00:24:12.180
Kathy Yelick: And this is just so you know, people often complain about the top 500 lists and I sometimes complain about the top 500 list. I think the best

147
00:24:12.840 --> 00:24:19.590
Kathy Yelick: Is a, it's a better reflection at this point of what deep learning algorithms look like, then it is especially compositional neural nets specifically

148
00:24:20.040 --> 00:24:32.040
Kathy Yelick: Than it is and most modeling and simulation problems even certainly doesn't reflect data analysis problems, but from a historical standpoint is actually quite interesting. So you can see over time how the, you know, vector supercomputers.

149
00:24:32.610 --> 00:24:44.370
Kathy Yelick: died out. And then we had shared memory and the supercomputers and massively parallel machines and then clusters of various kinds. And then the accelerated machines is purple where we see that really picking up

150
00:24:44.850 --> 00:24:52.230
Kathy Yelick: And of course, very different problems in terms of how hard it is to paralyze these. So the vector supercomputers, we could

151
00:24:53.100 --> 00:25:02.400
Kathy Yelick: Basically paralyzed by annotating cereal programs. We had to completely rethink the algorithms and the software. When we got to distributed memory. And unfortunately, I think.

152
00:25:02.700 --> 00:25:09.540
Kathy Yelick: A lot of that is happening, not necessarily between the nodes, but within the node again as we look at these accelerator architectures.

153
00:25:10.830 --> 00:25:23.280
Kathy Yelick: And this is just a picture of the different types of accelerators that are in the systems, you can see that over 100 about 150 of the systems as of 2019 had accelerators. I think this year it's even even higher.

154
00:25:25.260 --> 00:25:30.690
Kathy Yelick: And so what does this mean to me and I'd actually put together a slide about this, about 10 years ago and

155
00:25:31.380 --> 00:25:38.550
Kathy Yelick: And one of the main points still holds, which is we see a lot more data need for data parallelism within the GPU architecture.

156
00:25:39.210 --> 00:25:45.690
Kathy Yelick: Even though they think of it as threads. It's really, I think, very useful to think about data parallel algorithms, when you're trying to map on two

157

00:25:46.170 --> 00:25:59.550
Kathy Yelick: GPUs, a lot more memory spaces than I would ever have wanted to have. So a GPU memory space of CPU memory space. And oftentimes, not always memory meant a hardware managed levels of the memory hierarchy.

158
00:26:00.990 --> 00:26:12.990
Kathy Yelick: Perhaps one of my most frustrating parts to me are still in the case on the right, which is that the CPUs are in control. So the accelerators. The GPUs tend to only execute things when they're launched

159
00:26:13.410 --> 00:26:28.140
Kathy Yelick: From the CPU and that the CPUs are the ones responsible for the communication, for the most part, and that's something that we're really pushing on within the extra by on projects for reasons that you'll hopefully understand when we look at the algorithms. And so the

160
00:26:30.390 --> 00:26:33.930
Kathy Yelick: But the, this is not sort of fundamental to them. And I think that

161
00:26:34.170 --> 00:26:45.300
Kathy Yelick: If you look at something like the summit machine at Oakridge and certainly any plans for access scale machines. You've got well over 90% 95 99% event of your computing capability and the GPUs.

162
00:26:45.540 --> 00:26:51.090
Kathy Yelick: And the idea that you're putting the CPUs in control and treating the GPU as a co processor

163
00:26:51.540 --> 00:27:01.500
Kathy Yelick: Is actually I think quite an efficient. I think you want to think about running everything on the GPU and only rarely those pieces of the code that won't run on the GPU to run in the CPU.

164
00:27:02.310 --> 00:27:11.640
Kathy Yelick: So this is what we're trying to do, as I said, we didn't really start the extra by on Project. The idea that we were going to be focused on GPUs. But now that that is the architecture.

165
00:27:11.910 --> 00:27:18.270
Kathy Yelick: That it seems we will be using where we're trying to and finding surprising ways of mapping. Some of the algorithms onto GPUs.

166
00:27:19.410 --> 00:27:28.710
Kathy Yelick: So one of the things though we're in this idea of trying to put the accelerators in charge that is the GPUs and charging and in particular in charge of communication.

167
00:27:29.250 --> 00:27:34.380
Kathy Yelick: This was some work done recently by Taylor Barnes and others at nurse with

168
00:27:34.980 --> 00:27:50.700
Kathy Yelick: The people in my group, looking at trying to do direct communication from a GPU into the network. So I'm sure many of you are familiar with envy link, which is a an NVIDIA systems. It's our communication conduit between the GPUs and

169
00:27:51.930 --> 00:28:04.290
Kathy Yelick: On the summit system at Oak Ridge National Labs because Nvidia worked closely with IBM IBM put on the power nine architecture and interface for the for the NB link.

170
00:28:05.520 --> 00:28:12.660
Kathy Yelick: Our interface communication interface so that the GPU and the CPU really can talk much more directly in that protocol, then

171
00:28:13.230 --> 00:28:20.820
Kathy Yelick: Then they can and most other CPU, GPU pairs. So this is kind of a best case. But even on this system. And I think part of this is software.

172
00:28:21.120 --> 00:28:29.490
Kathy Yelick: At this point, not necessarily fundamental to the hardware when we try to initiate communication on a GPU rather than initiating and CPU.

173
00:28:29.850 --> 00:28:41.790

Kathy Yelick: You see really significant synchronization overheads. So it almost seems that what's really happening is, although you're trying to initiate communication from directly from the GPU and here we're doing a one sided our DMA

174
00:28:42.450 --> 00:28:45.150
Kathy Yelick: Communication, which should be about the fastest thing you could do.

175
00:28:45.450 --> 00:28:54.450
Kathy Yelick: On either side and you see that that really is still basically coordinating with the CPU and there's these things about ringing the doorbell and so on, that the CPU.

176
00:28:54.690 --> 00:29:03.000
Kathy Yelick: ends up having to do so we're not really able to get competitive performance by direct GPU communication. And on the right hand side you can see what happens with

177
00:29:03.270 --> 00:29:17.610
Kathy Yelick: With bandwidth or time per bites and, you know, although eventually they get to sort of pretty close to the same point. You need a much larger message on the GPU in order to be able to get the same kind of bandwidth performance at the system.

178
00:29:19.050 --> 00:29:30.000
Kathy Yelick: And now, more broadly, looking at the architectures. I'm sure you are all all know that the most expensive thing you do in any of these systems is move data around. So communication of various kinds.

179
00:29:30.330 --> 00:29:36.990
Kathy Yelick: This is a graph put together by Jim demo, but based on data from the latest edition of the Hennessy Patterson architecture book.

180
00:29:37.470 --> 00:29:47.160
Kathy Yelick: So this is looking at network latency here is the slowest thing on the graph. The network bandwidth in terms of the time per, per, per bite and

181
00:29:47.730 --> 00:29:53.970
Kathy Yelick: And the or per word. I think that is actually and then the memory latency and the communication latency and of course down here.

182
00:29:54.510 --> 00:30:05.070
Kathy Yelick: Is the floating point throughput and the this with this gamma term. So the total time to then, you know, just a simple formula to say the time to compute everything is the

183
00:30:05.700 --> 00:30:12.870
Kathy Yelick: All the floating point operations, times the time for flop. By the way, in the genomics applications, many of them don't do any floating point the assembly problem really does.

184
00:30:13.770 --> 00:30:21.480
Kathy Yelick: Know floating point, but you can pick a different integer or logical operation. But the point is still the same. That, of course,

185
00:30:22.110 --> 00:30:32.310
Kathy Yelick: Although back in the 80s, the expensive thing was computing floating point. Now it's it's definitely at least on a single node system. Now it's definitely the data movement.

186
00:30:33.990 --> 00:30:41.250
Kathy Yelick: So the other thing that's changed since that 2007 report is that the science community. So it's not just that you know the

187
00:30:41.580 --> 00:30:50.280
Kathy Yelick: The rest of the world has gotten involved in machine learning, but that the rest of the science community has also gotten very interested in much larger scale data analysis problems.

188
00:30:50.580 --> 00:31:00.360
Kathy Yelick: Basically all of the instruments, the scientific instruments, whether it's light sources genome sequencers if you're out analyzing even simulation data these data sets have become

189
00:31:00.780 --> 00:31:09.510
Kathy Yelick: Very large and so there's there's kind of basic data analysis problems that you want to do on it. And then, of course, people want to also do machine learning, I'll probably say

190
00:31:09.960 --> 00:31:23.670
Kathy Yelick: Although I we are working on that within the IBM project. I'm going to say a little bit less about that today except as I mentioned the the clustering algorithm. But we're using things like deep learning algorithms to try to analyze some of this meta genome data as well.

191
00:31:25.200 --> 00:31:38.040
Kathy Yelick: So a report that I was involved with them last year was looking at what's called AI for science. The word AI here is really interpret is being used much more broadly to include a data analysis.

192
00:31:38.460 --> 00:31:45.930
Kathy Yelick: Algorithms are various kinds all different types of statistical analysis and machine learning, not just not just deep learning and but

193
00:31:46.560 --> 00:31:57.930
Kathy Yelick: These are the we. There were three town halls that were run by each of the three science computing labs are gone Oakridge and Berkeley and the breakout groups had a

194
00:31:58.530 --> 00:32:04.710
Kathy Yelick: Looked at a lot of the different science problems as well as some of the cross cutting themes. And so there's a report available if you're interested in seeing that

195
00:32:05.160 --> 00:32:15.330
Kathy Yelick: And the asked kak subcommittee which advises do we on this has a report a draft that's that was out as of a few weeks ago, I think they're just finalizing that getting comments back so

196
00:32:15.570 --> 00:32:21.090
Kathy Yelick: I think there is a big shift happening in in the Department of Energy, but more broadly in the science community.

197
00:32:21.630 --> 00:32:27.150
Kathy Yelick: Away from just using high performance computing for simulation problems and using it for data analysis problems.

198

00:32:27.450 --> 00:32:38.490
Kathy Yelick: And machine learning problems and I separate those because some of these problems. There's a lot of data analysis problems that are not machine learning inserted before you get to the point of machine learning and others that that are

199
00:32:39.600 --> 00:32:50.280
Kathy Yelick: Machine learning. OK. So now on to the algorithms. So I put together a paper, a few months ago or last year on just came out in January on

200
00:32:51.120 --> 00:33:04.200
Kathy Yelick: Some of the computational patterns that come up in genomic analysis. And what was interesting to me, and I'll say a little bit more about this is how similar it was to another set of kind of computational motifs for big data problems that the National Academies.

201
00:33:05.190 --> 00:33:12.300
Kathy Yelick: Committee, put one of their, their committees put together. So these are the algorithms that we see a lot in

202
00:33:13.290 --> 00:33:23.220
Kathy Yelick: In these genomics problems we see a lot of hashing and hash tables sorting. I won't say so much about sorting, but it's kind of an alternative to hashing for many of these problems graphs.

203
00:33:23.580 --> 00:33:31.770
Kathy Yelick: A string alignment problems of various kinds, what I'll call generalized and body are they called they called generalize and body which is sort of an ultra all sort of computation.

204
00:33:31.980 --> 00:33:43.410
Kathy Yelick: And then both dance and sparse matrices. And these are some of the applications that I've already mentioned, but it comes up and things like annotating genes, trying to figure out what their functional behavior is and things like that as well.

205
00:33:45.210 --> 00:33:45.660
Kathy Yelick: So,

206
00:33:46.830 --> 00:33:56.670

Kathy Yelick: If you're familiar with the The seven dwarfs of scientific computing. It was a phrase that Phil Colella put together or used over a decade ago.

207
00:33:57.270 --> 00:34:03.990
Kathy Yelick: And his original list which I actually still go back to I like the original set of things which really come from scientific computing

208
00:34:04.320 --> 00:34:09.930
Kathy Yelick: I think still reflect a lot of the computational patterns that come up in scientific applications.

209
00:34:10.440 --> 00:34:15.900
Kathy Yelick: What's interesting, this was the the for that from the National Academies report have these seven giants of big data.

210
00:34:16.680 --> 00:34:25.410
Kathy Yelick: And some of these kind of particle methods and generalized and body. There's some similarities, although there's also some significant differences and I'll say a little bit more about that one.

211
00:34:26.430 --> 00:34:38.550
Kathy Yelick: Graph theory and sort of graph algorithms. I think that was is a little bit broader than what they meant in that particular National Academies report come up a lot in in this genomics space and then

212
00:34:39.900 --> 00:34:49.650
Kathy Yelick: These integration and optimization algorithms. I find a little bit less useful as sort of understanding the parallelism patterns. I think that these these problems like hashing and sorting are really important though.

213
00:34:50.970 --> 00:34:53.070
Kathy Yelick: So that's what I would replace them with

214
00:34:54.420 --> 00:35:06.300

Kathy Yelick: Now, as many of you know I've worked for most of my career on these class of algorithms called partition global address based languages and and I'm not going to go into much detail on them, except to say that

215
00:35:06.840 --> 00:35:09.480
Kathy Yelick: It really gives you a different view of a distributed memory.

216
00:35:10.320 --> 00:35:15.990
Kathy Yelick: System. And I think one that perhaps we still need a better algorithmic model for thinking about, which is

217
00:35:16.260 --> 00:35:26.490
Kathy Yelick: It's not really both synchronous and it doesn't require a synchronous send receive or even an asynchronous and received is the ability to read and write memory anywhere in the system.

218
00:35:26.850 --> 00:35:41.460
Kathy Yelick: From any one of the processors. So one sided communication remote put and remote get and you'll see why this is useful in some of the algorithms that come up an ex IBM. So at this point, where my my work is mostly focused actually in this microbial

219
00:35:42.090 --> 00:35:51.090
Kathy Yelick: Data Analysis space and originally I got into it because it looked like this. General Assembly problem was a good fit for this, these, these

220
00:35:51.900 --> 00:35:57.060
Kathy Yelick: Languages UPC at the time. Although now rewritten it up C plus plus, but

221
00:35:57.690 --> 00:36:05.490
Kathy Yelick: And the ability to have these global pointers that any processor can point to any other processors memory segments. And then I'm doing a remote reader. Right.

222
00:36:05.730 --> 00:36:16.740
Kathy Yelick: Is I think a powerful way of looking at certain classes of algorithms, not all of them, but things that are very irregular and have sort of a random access characteristics such as building a hash table.

223
00:36:17.910 --> 00:36:20.430
Kathy Yelick: So in the genome assembly.

224
00:36:21.450 --> 00:36:26.610
Kathy Yelick: What we do is we take in these Reed's we chop them into fixed length strings called cameras.

225
00:36:27.450 --> 00:36:33.870
Kathy Yelick: And we then we histogram those cameras. We do some analysis on and we actually throw out the ones that only occur.

226
00:36:34.470 --> 00:36:43.440
Kathy Yelick: What a single time because those are probably errors we then build a double line graph from those and and then walk through that to Brian graph and

227
00:36:44.010 --> 00:36:48.600
Kathy Yelick: Could use that to compute the connected components in that graph which are the context so

228
00:36:49.230 --> 00:36:55.140
Kathy Yelick: The way the cameras aren't analyzed we we keep the, the left character on the right character for each one of the

229
00:36:55.500 --> 00:37:00.930
Kathy Yelick: Cameras we store that in a hash table and then you can look at that hash table is a graph and walk through that graph.

230
00:37:01.170 --> 00:37:15.330
Kathy Yelick: To find the context. And during this camera analysis phase, we can also throw out, for example, any, any ambiguities so that what we end up with are very highly we're very confident of these fragments of the final genome.

231
00:37:15.840 --> 00:37:21.810

Kathy Yelick: But there, they tend to still be pretty fragmented. They're not not very complete they're much more complete than the original reads are

232
00:37:22.830 --> 00:37:27.120
Kathy Yelick: But then we tried to extend them by aligning the original reads back to them.

233
00:37:27.540 --> 00:37:35.370
Kathy Yelick: Because we've been we've been pretty conservative in that that content generation step and that requires an imperfect string alignment.

234
00:37:35.730 --> 00:37:47.970
Kathy Yelick: And which is quite expensive. And then we use that to try to build something called scaffold. And this is done with another graph walk algorithm that I won't really talk more about but is another distributed memory case in this

235
00:37:48.600 --> 00:38:03.060
Kathy Yelick: graph algorithm in this particular case we can often distribute the graph on in a way that most of the, we can do a partitioning on it. For example, and and that that graph walk is not as communication intensive is some of the other parts of the code.

236
00:38:04.860 --> 00:38:18.480
Kathy Yelick: So originally this code was written in a combination of two languages MPI for libraries for the camera analysis, I'd village had written the camera analysis part of it, and he tends to write things in bulk synchronous MPI

237
00:38:18.990 --> 00:38:26.610
Kathy Yelick: With collectives and and then the, the rest of the code is written by evangelists here ganas as part of his PhD thesis and was all written in UBC

238
00:38:26.910 --> 00:38:40.020
Kathy Yelick: This was a bit of a problem for us because it was, it takes up a lot of memory to hold both the MPI and UPC runtime at the same time. So at this point, it's all been now rewritten into up C plus plus, including that content generation piece.

239
00:38:41.250 --> 00:38:51.480

Kathy Yelick: And just to give you an idea, this actually does scale across multiple nodes. This is the original UPC code, but as scaling to over 1000 nodes so many more cores.

240
00:38:52.170 --> 00:39:00.630
Kathy Yelick: And and you can see the different pieces of it that are scaling pretty well scaffolding actually was one of the pieces that was not scaling as well.

241
00:39:00.930 --> 00:39:15.840
Kathy Yelick: And this had to do with load balance because as I said, we can partition the graph pretty well, but there are some pieces of that graph that connected components are actually quite large and so that that's where we have to do some more work in the in the scaffolding phase. Whoops.

242
00:39:17.970 --> 00:39:18.210
Sorry.

243
00:39:20.790 --> 00:39:33.510
Kathy Yelick: I'm still there. Right. Okay, so a camera analysis, we take the reason we chop them into cameras. They're sliding windows. So at every single character position you compute another kaymer we then build a histogram of those and

244
00:39:34.770 --> 00:39:40.200
Kathy Yelick: We also use a well anyway so so we take these we're building these these

245
00:39:41.700 --> 00:39:46.200
Kathy Yelick: Histogram. These kaymer histograms by hashing. The kaymer and then

246
00:39:46.860 --> 00:39:56.430
Kathy Yelick: Computing its value. But this is going to be a distributed memory data structure because if you think about it, kind of on the surface, the size of the dataset grows by almost a factor of k

247
00:39:57.090 --> 00:40:02.100
Kathy Yelick: Given some some things like k because you're taking every string and making kind of a key length copy of

248
00:40:02.640 --> 00:40:13.800
Kathy Yelick: It at every position you know module of the the tail at the end when the but but for short gamers. It's basically k times larger. So the data becomes quite large and doesn't fit in a single, shared memory node.

249
00:40:14.310 --> 00:40:27.480
Kathy Yelick: So we will build this in distributed memory. So this is built in with also with a bloom filter which we do to save memory. So the first thing is to build this Bloom filter. I see. There's my mouse.

250
00:40:28.530 --> 00:40:33.900
Kathy Yelick: So we distribute the bloom filter over processors, we end up having though to communicate all of the cameras.

251
00:40:34.260 --> 00:40:41.550
Kathy Yelick: And there is an optimization for heavy hitters that's in the code that is useful for certain genomes, depending on how much repetitiveness there is

252
00:40:42.060 --> 00:40:48.420
Kathy Yelick: In the set of cameras in the genome. But, but, in general, everything all the cameras get sent all around the machine.

253
00:40:48.690 --> 00:40:55.650
Kathy Yelick: And then we build a bloom filter and use that to instantiate the hash table. So we the bloom filter filters out the singleton's

254
00:40:56.220 --> 00:41:04.560
Kathy Yelick: That saves us about maybe a factor of two or so and memory doesn't really save us and running times you'll see. But then we do it all over again with the distributed hash table.

255
00:41:05.370 --> 00:41:10.350
Kathy Yelick: And this is just some scaling results from the camera analysis and hammer

256
00:41:10.950 --> 00:41:21.660

Kathy Yelick: The hammer camera analysis then when it builds the hash table it's storing two characters, the left and the right extension, as I said, the where what appeared in the original read before and after that kaymer

257
00:41:22.050 --> 00:41:32.010
Kathy Yelick: So a little bit less data is, you know, not very much data is being communicated along with the kaymer scales pretty well, though not perfectly as you scale up to 4K nodes are so

258
00:41:33.150 --> 00:41:41.280
Kathy Yelick: A second assembler that we're building called D Bella also does camera analysis and I'll mention a little bit more about it later. But it's a

259
00:41:42.330 --> 00:41:50.790
Kathy Yelick: It has a much higher payload that is in addition to the camera. You're also keeping track of the ID that they came or appeared in the read ID and also

260
00:41:51.060 --> 00:41:57.930
Kathy Yelick: The position within the read. And so the communication volume is higher, and therefore the communication cost makes a little bit less scalable.

261
00:41:58.650 --> 00:42:06.330
Kathy Yelick: At that point, although the computation that will follow this and the develop cases very large. And so, as you'll see, will be able to hide that communication.

262
00:42:08.820 --> 00:42:09.300
Kathy Yelick: So,

263
00:42:10.680 --> 00:42:25.740
Kathy Yelick: Now that the camera County and from the original code that was in a box synchronous MPI model is now in UPC plus plus. And this just shows the running time of the PC plus plus code is actually faster than the MPI code.

264
00:42:26.550 --> 00:42:33.660
Kathy Yelick: And and the with the the up c++ code, you also seen it without the bloom filter where it's actually faster.

265
00:42:34.290 --> 00:42:46.500
Kathy Yelick: You know, we use the bloom filter. When we need the memory, but it's but it's faster to not do the second because it's a complete completely separate round. You do have to to do you know the communication twice.

266
00:42:47.850 --> 00:42:58.230
Kathy Yelick: We are looking at GPU optimizations for camera counting, you might get a sense of why we really care about being able to communicate from the GPU. So this is looking at

267
00:42:59.010 --> 00:43:03.180
Kathy Yelick: Some speed ups of just doing camera counting on a single GPUs on a

268
00:43:03.930 --> 00:43:11.760
Kathy Yelick: On a Tesla V 100 so we can get some nice speed ups, although even on the single GPU. We're actually spending more of our time.

269
00:43:12.060 --> 00:43:27.540
Kathy Yelick: Communicating so this, this is the GPU breakdown. How much time that GPUs busy and the little blue boxes here so you know point 09 and point 04 out of 1.6 so most of the time is actually spent sending the cameras to the GPU so that they can be

270
00:43:28.320 --> 00:43:34.680
Kathy Yelick: Counted and so we're actually now sending the RAW, reads and parsing and things on the GPU in order to try to

271
00:43:34.980 --> 00:43:46.500
Kathy Yelick: amortize that but you can see why we don't want to go back to the CPU in order to send every one of these cameras or at least p minus one over p of the cameras typically is going to go to another processor. We don't want to send them.

272
00:43:47.370 --> 00:43:51.780
Kathy Yelick: Through the CPU by by going back that that through that that channel.

273
00:43:54.120 --> 00:44:05.070

Kathy Yelick: So the next thing that happens in the after kaymer counting and camera analysis is the graph walk. This is a paragraph reversal. So once we've got our hash table.

274
00:44:05.310 --> 00:44:14.370
Kathy Yelick: And what you can see in here is a little three more. So a key like a TC and the left and right extension t and g where the characters that appeared before and after that the original reads

275
00:44:15.120 --> 00:44:21.390
Kathy Yelick: And then what we do at the algorithm is just picks up a random seed that a random one of these cameras.

276
00:44:21.900 --> 00:44:34.080
Kathy Yelick: In the hash table and then starts walking through the graph, both to the left and then to the right and there is a non trivial synchronization protocol that happens because multiple processors can run into the same

277
00:44:34.530 --> 00:44:40.080
Kathy Yelick: The same kaymer and therefore have to figure out how to avoid walking through the same parts of the graph.

278
00:44:40.800 --> 00:44:45.720
Kathy Yelick: And but this is just a little bit, you know, gives you an idea of how that works. Each one of these, of course.

279
00:44:46.020 --> 00:44:58.440
Kathy Yelick: On average, you're going to be doing a remote lookup for each one of these. So you would think this wouldn't scale well at all. But what happens is because all of the as I showed you the scale a little scaling results before it scales pretty well because

280
00:44:59.820 --> 00:45:12.570
Kathy Yelick: Basically have all the processors doing all of this message injection. So you get some slow down, going from one processor to two, but after that you actually get quite quite efficient or one node to two, but you get very efficient scaling.

281
00:45:14.460 --> 00:45:23.490

Kathy Yelick: And so you can have multiple processors doing it, as I said, they do have to coordinate because they can easily run into the same the same cameras, they they walked through the these vertices. So

282
00:45:24.000 --> 00:45:31.110
Kathy Yelick: Call this a hash table, call it a graph that's that's how it but that's what the algorithm is actually doing is walking through it by looking things up in a hash table.

283
00:45:32.550 --> 00:45:37.290
Kathy Yelick: One of the things that evangelist did, and part of his thesis was to look at better ways of hashing.

284
00:45:37.770 --> 00:45:40.860
Kathy Yelick: So that you can improve locality and reduce the amount of communication.

285
00:45:41.190 --> 00:45:47.430
Kathy Yelick: And the answer is, well, I have no idea what my genome looks like until I've assembled it so I don't know what these contexts are

286
00:45:47.640 --> 00:45:55.020
Kathy Yelick: until I've walked through my hash table. So there's not really a good way of doing this, but there's a few use cases that actually come up where you can come up with an oracle

287
00:45:55.710 --> 00:46:05.730
Kathy Yelick: For example, if you did have an oracle that told you how to what the hash function was that would give you a good locality to put each one of these contexts, these sets of

288
00:46:06.720 --> 00:46:11.850
Kathy Yelick: Cameras on a different processor than you could avoid a lot of the communication and so

289
00:46:12.270 --> 00:46:17.580
Kathy Yelick: Some things that come up in practice is that you're assembling a species that you've seen before, such as the human genome or

290

00:46:17.760 --> 00:46:26.700
Kathy Yelick: When I mentioned that pan genome example. So after you've done the first assembly. You can use that to to give you a hash function that will do a pretty good job of laying out the

291
00:46:27.420 --> 00:46:35.340
Kathy Yelick: The reeds, actually, and the end then are the the cameras in the in the hash the camera phase based on

292
00:46:35.850 --> 00:46:48.420
Kathy Yelick: We and we do both of them. We try to both optimize the layout and also the the camera hashing to give you better locality so that even though the first jump might be remote, then you'll, you'll be end up being on the same processor for a while.

293
00:46:49.950 --> 00:47:02.250
Kathy Yelick: And also the other places comes up in our meta genome example is we're going to use an iterative. I didn't really mention that when I talked about meta hit more but you actually run through that whole pipeline multiple times using different values of k

294
00:47:02.580 --> 00:47:16.020
Kathy Yelick: And so you can use that. And so that gives you some some reasonable partitioning of these, and that makes things much happier. In practice, this gives us a factor of almost three speed up in terms of the graph reversal time and a

295
00:47:17.100 --> 00:47:23.940
Kathy Yelick: New save you over 770 6% of the off node communication. And when you do this for a particular input.

296
00:47:25.560 --> 00:47:31.170
Kathy Yelick: Now the last sort of algorithm that I think I'll talk about in any detail is alignment and

297
00:47:31.680 --> 00:47:37.710
Kathy Yelick: Alignment comes up in actually all the, all the different pieces of the Exabiome project. So in Whitmer,

298

00:47:37.890 --> 00:47:49.440
Kathy Yelick: After we've computed these contexts. We're going to read a line. The original reads to that. And we do that, you know, the difference is that this is now imperfect string alignment where we're allowing insertions, deletions and mismatches substitutions.

299
00:47:50.460 --> 00:48:03.000
Kathy Yelick: We often do this if we're comparing to a reference. We actually have somebody that wants to use our liner for some of the coven data. One of the things that they do with the coven say the nasal sample which is a microbial sample.

300
00:48:04.170 --> 00:48:09.390
Kathy Yelick: Is they want to filter out all of the human DNA so that they don't leak any di

301
00:48:10.350 --> 00:48:20.100
Kathy Yelick: D identify our identity identifiable data about human DNA from the the samples. And so there's a fairly expensive filtering process that is alignment against a fixed reference

302
00:48:20.670 --> 00:48:36.000
Kathy Yelick: In our second assembler, it actually doesn't do a brain graph assembly. It actually just aligns all of the RAW, reads to each other. This comes up and on different sequencing technology these long reads and I'll say a little bit more about why that comes up.

303
00:48:37.140 --> 00:48:49.560
Kathy Yelick: So without going into detail alignment is a dynamic programming problem. So worst case order N squared algorithm defined what is the optimal path through these two strings that will get them to align

304
00:48:50.610 --> 00:48:59.160
Kathy Yelick: In practice, we often want to align many things too many things, or at least many things to one thing. So the as an example I showed before. So what we'll do is we will

305
00:48:59.340 --> 00:49:10.290
Kathy Yelick: Only align things that have at least one identical kaymer and that filtering allows us to cut down substantially on the order n square that happens before you get to the actual string alignment algorithm.

306

00:49:10.560 --> 00:49:15.690
Kathy Yelick: Is called a seat and extend algorithm. And it can be done in a number of different ways. We're doing it based on these

307
00:49:15.930 --> 00:49:23.130
Kathy Yelick: These cameras. So once again, we're going to use this hash table now for a different purpose, which is to find, for example, the context that map.

308
00:49:23.460 --> 00:49:29.550
Kathy Yelick: That that are going to match a particular read and we we build that hash table and then we look them up.

309
00:49:29.820 --> 00:49:38.370
Kathy Yelick: Look up the the cameras that are in all the reads in order to figure out which reads to align to which context. And then we run the dynamic programming algorithm on that on those pairs.

310
00:49:38.790 --> 00:49:48.420
Kathy Yelick: And by the way, there are many variations of these. I know there's a lot of papers out there on parallel Smith Waterman GPU optimized with Waterman etc. So with Waterman being one of the

311
00:49:49.170 --> 00:49:57.270
Kathy Yelick: Canonical the original algorithm for genome will actually need them and lunches, the original. But anyway, those to close variations of each other.

312
00:49:57.960 --> 00:50:05.910
Kathy Yelick: Many of the algorithms that we want to use in practice will will, for example, terminate early if it looks like the score is very poor. It cuts off the upper

313
00:50:06.600 --> 00:50:15.180
Kathy Yelick: Upper right and the lower left part of the search space because something that is aligning down in those regions is probably not going to be a very good alignment.

314
00:50:15.360 --> 00:50:29.520

Kathy Yelick: And we only care about fairly high quality alignments. And there's a lot of other heuristics that come up in practice to speed this up. And so I think there's actually is an opportunity for domain specific languages because there's not just one optimized.

315
00:50:31.020 --> 00:50:40.410
Kathy Yelick: You know alignment algorithm. But they're all very closely related. We also by the way one are aligned proteins as well as DNA so proteins have have a much larger alphabets.

316
00:50:41.190 --> 00:50:50.460
Kathy Yelick: As opposed to just the four characters in DNA but the algorithms are essentially at some level, the same and similarly for the hash tables we we've got hash tables.

317
00:50:50.970 --> 00:50:58.140
Kathy Yelick: Many different hash tables throughout the assemblers and and actually through some of the other parts of the application as well.

318
00:50:58.560 --> 00:51:07.770
Kathy Yelick: So a little bit of, you know, can you can you optimize GPU alignment on GPUs. Yes, you can. And this was some work actually done in a class project.

319
00:51:08.100 --> 00:51:16.770
Kathy Yelick: A couple of years ago, but the x axis here is the length of the string that you're aligning. And so what you're seeing here is the GPU time in green, and then the

320
00:51:17.340 --> 00:51:21.570
Kathy Yelick: The original sequential time and black. And there's an open MP and a shared memory SMP node.

321
00:51:21.930 --> 00:51:30.420
Kathy Yelick: So what you can see as you get substantial speed ups but when the length is out here in the 10 10,000 characters and many of the things that were read we're aligning our

322
00:51:30.690 --> 00:51:38.490
Kathy Yelick: The reeds for the short read case. So in some part of the alumina reads about 150 base pairs. So we're way over here where you're not really getting much speed ups.

323
00:51:38.760 --> 00:51:49.980
Kathy Yelick: I'm instead, what we're doing is, then, of course, doing multiple alignments together as a single group a batch alignment, if you will. And this is some work done by was the one looking at

324
00:51:50.520 --> 00:52:00.060
Kathy Yelick: The speed up that you get from doing that. But once again, you know, we don't want to just do these things on a single GPU, we have to put them into this distributed memory framework.

325
00:52:00.540 --> 00:52:11.790
Kathy Yelick: We've done that. Just recently, these are brand new results and actually from this morning, I think. So this is speeding up the alignments to the alignment phases of the metal hammer pipeline.

326
00:52:12.330 --> 00:52:21.600
Kathy Yelick: So we're not trying to GPU fit the other parts of it yet although we've got a Kamer that I showed you a little bit on camera analysis, but that's not yet in this code.

327
00:52:22.080 --> 00:52:29.970
Kathy Yelick: But you can see we we get sort of decent speed ups on the alignment pieces of it. But there's still a lot of distributed memory communication happening.

328
00:52:30.210 --> 00:52:42.060
Kathy Yelick: And so we would really like to be able to do efficient one sided communication from the directly from the GPUs and, you know, my little cartoon here is the fact that this is kind of now a game of GPU whack a mole where we're going to try to

329
00:52:42.450 --> 00:52:54.420
Kathy Yelick: There was not not one phase that completely dominated the computation here in the long read assembler, actually there's a lot more time spent in just pure alignment. So I think they're the GPUs will be much more have a much more immediate impact.

330
00:52:55.470 --> 00:53:06.300
Kathy Yelick: So on the long Rita liner, just briefly, you know, what's different about long reads well there longer so they may be over 10,000 base pairs as opposed to say 150 base pairs.

331
00:53:06.900 --> 00:53:10.200
Kathy Yelick: So the problems are more compute intensive and they're more GPU friendly.

332
00:53:10.770 --> 00:53:15.360
Kathy Yelick: We don't necessarily use a divine graph, although there's still some debate in the community about that.

333
00:53:15.630 --> 00:53:20.250
Kathy Yelick: And we just do these pairwise alignments. And you can think of it as an embarrassingly parallel all to all

334
00:53:20.460 --> 00:53:31.740
Kathy Yelick: Alignment problem, this sort of generalize and body, if you will, but you really want to do is first at alignments. You want to only use things that have a common kaymer in them. And in fact, we also filter the cameras.

335
00:53:32.160 --> 00:53:36.420
Kathy Yelick: Because some of them occur with such high frequency. And that's what the graph on the right is showing

336
00:53:36.840 --> 00:53:51.420
Kathy Yelick: That they're probably going to give you a lead to a bunch of spurious overlaps and therefore a lot of cost in terms of the alignments. And so we we filter them out as well. This turns into a sparse matrix problem that is the kaymer by read sparse matrix which we multiply times the

337
00:53:52.710 --> 00:54:00.510
Kathy Yelick: Sea, they read by camera times the camera by reads it's transpose and get a read by read sparse matrix out and that tells you which ones have the

338
00:54:00.810 --> 00:54:13.110
Kathy Yelick: Pairs of reads to align. So lots of opportunities here and for for both distributed memory optimizations communication avoiding algorithms and and also on GPU optimizations.

339

00:54:14.490 --> 00:54:22.860
Kathy Yelick: So this is just a little bit of a time breakdown, which I won't say talk about in detail, but like the other liner. Like the other assembler those. This is for the long read assembler

340
00:54:23.490 --> 00:54:34.380
Kathy Yelick: And there are a number of different phases. So we're going to have to put GPU optimizations into the camera analysis and and everything else inside of here, we did recently.

341
00:54:35.130 --> 00:54:38.880
Kathy Yelick: marquis to Alice. Just finished her PhD thesis, looking at a synchrony.

342
00:54:39.060 --> 00:54:47.310
Kathy Yelick: Versus box synchrony in this this long rate assembler in this part of the overlap or. And so what you can see from this graph is that she can completely hide.

343
00:54:47.880 --> 00:54:51.210
Kathy Yelick: The communication time using the one side communication.

344
00:54:51.690 --> 00:55:00.330
Kathy Yelick: She's actually also analyzed the memory footprint and shows that the asynchronous algorithm that is using up C plus plus, and one sided communication just sort of on demand.

345
00:55:00.690 --> 00:55:16.050
Kathy Yelick: Is also use a much lower memory footprint, at least on most most configurations until you get to the largest one here. So there's certain advantages. What you can see as left here is some synchronization time. So there's some load imbalance that probably still needs to be addressed.

346
00:55:17.250 --> 00:55:22.020
Kathy Yelick: I think I'll skip this since i said i think i hadn't talked a little bit more hopefully about hit more

347
00:55:22.470 --> 00:55:29.340
Kathy Yelick: But it also he turns everything into a sparse matrix algorithm, although you can think of it as a graph algorithm. And I'll just wrap up by saying that

348
00:55:29.970 --> 00:55:38.700
Kathy Yelick: In science, I think that simulation problems have not gone away, and there's still a number of simulation problems that are too expensive for

349
00:55:39.840 --> 00:55:44.730
Kathy Yelick: For single node system certainly and require much larger scale HTC systems.

350
00:55:45.180 --> 00:55:53.190
Kathy Yelick: But they're also a lot of data analysis problems now because the data sets have grown and certainly they've grown in science in a different in addition to growing in

351
00:55:53.460 --> 00:55:59.910
Kathy Yelick: Business and other applications that maybe you're more familiar with. And there are a lot of machine learning problems that are both

352
00:56:00.630 --> 00:56:14.400
Kathy Yelick: Too large and too expensive. And we're so we're using our HTC systems for that as well. And, but I think that we see a somewhat different set of computational patterns that come up here, not just the lack of floating point. But really, that

353
00:56:15.060 --> 00:56:31.110
Kathy Yelick: We see a lot more of this irregular memory access kind of workload that comes from things like hash tables and sparse matrices that are very unstructured rather than the kinds of sparse matrices that arise in simulation problems. So with that, I will stop and happy to answer any

354
00:56:31.110 --> 00:56:34.980
Julian Shun: Questions. Oops. Excellent, excellent. Kathy.

355
00:56:35.850 --> 00:56:37.380
Julian Shun: Do a virtual. APPLAUSE

356
00:56:40.590 --> 00:56:46.740
Julian Shun: So if anyone has any questions, please feel free to unmute yourself and ask

357
00:56:48.450 --> 00:56:58.980
Julian Shun: So I'll start off with a question. So I had a question regarding the portability of the algorithms. So I'm wondering if you need to.

358
00:56:59.370 --> 00:57:09.990
Julian Shun: Tune the algorithms for different types of machines and also for different types of data sets, or are you using basically the same code for different machines that data sets.

359
00:57:10.590 --> 00:57:11.640
Kathy Yelick: Yeah, so, so I can

360
00:57:12.360 --> 00:57:23.100
Kathy Yelick: answer that in a couple different ways. So from the machine standpoint, yes, we're going to use different code that I mean I think a DSL would be helpful in here for some of the underlying computational kernel is like the alignment and maybe the camera accounting

361
00:57:23.880 --> 00:57:34.350
Kathy Yelick: But we, we are not planning to use kind of a higher level language like open MP4 programming the GPUs on the different access scale architecture. So we've

362
00:57:34.590 --> 00:57:42.750
Kathy Yelick: We've got code right now and optimized for NVIDIA and we'll be in working starting to work on optimizing that for AMD GPUs, which will be in the Oakridge machine.

363
00:57:43.230 --> 00:57:50.880
Kathy Yelick: Haven't really started yet on the, the Intel GPUs, but because it's a fairly small set of kernels. We've decided just to and there

364
00:57:51.240 --> 00:58:01.860
Kathy Yelick: We decided to go ahead and optimize them for those different architectures. And I think there are different features of the architectures that are missing on some that will I think effect exactly how they get written

365
00:58:03.030 --> 00:58:11.040

Kathy Yelick: The, the communication and kind of the higher level stuff is all portable that's up c++ code written on gas now. So the gas net Communication layer has to be

366
00:58:11.460 --> 00:58:21.720
Kathy Yelick: Re optimized. I think exactly how if we get device level that is GPU to GPU communication that may be somewhat specialized to the architecture. So

367
00:58:22.800 --> 00:58:23.550
Kathy Yelick: Anyway, we're

368
00:58:25.620 --> 00:58:29.970
Kathy Yelick: You know, so, so it's different from that. But the other thing is that if you look at some of the

369
00:58:30.630 --> 00:58:41.400
Kathy Yelick: Library sort of naturally occurring libraries that right now are different instances we I mean we have refactor the code to use a single hash table implementation, but each one of those hash tables can be quite different.

370
00:58:41.670 --> 00:58:48.720
Kathy Yelick: And things like you know do Bloom filters work does a heavy hitters algorithm work. Those are really dependent on sometimes on the data set as well.

371
00:58:49.890 --> 00:58:50.550
Kathy Yelick: That's great question.

372
00:58:51.330 --> 00:58:52.560
Julian Shun: Great, thanks a lot.

373
00:58:55.770 --> 00:58:57.210
Julian Shun: Any, any other questions.

374
00:58:57.660 --> 00:58:58.020
Yes.

375
00:58:59.490 --> 00:59:00.300
Yiqiu Wang: I have a question.

376
00:59:01.230 --> 00:59:11.700
Yiqiu Wang: Yeah. Thanks for a great talk. So I'm not familiar with the field. I'm curious, the algorithms for instance the genome assembly printing clustering that did you already doing with static or evolving datasets.

377
00:59:13.770 --> 00:59:22.440
Kathy Yelick: Um, let's see. So I would say that the genome assembly problem is really static. We have a data set. There's a fixed data set that you're going to try to assemble so

378
00:59:22.620 --> 00:59:23.190
Kathy Yelick: You may

379
00:59:24.210 --> 00:59:35.400
Kathy Yelick: There are more and more data sets coming in over time, but you're not trying to update the assembly based on those new data sets, you've just got one, the one assembly for each each data sets.

380
00:59:36.540 --> 00:59:42.240
Kathy Yelick: That the protein clustering. Actually, that's it. That's a great question and protein clustering. It does incrementally change because

381
00:59:42.600 --> 00:59:49.620
Kathy Yelick: The question right now. What they do is if you've got a new update to the database. It's got a bunch of new protein Senate.

382
00:59:49.830 --> 00:59:57.030
Kathy Yelick: You actually start from scratch and rerun the algorithms all over again, which is which. You know, when it used to take them 15 weeks to do this was a bit of a problem.

383
00:59:57.660 --> 01:00:03.060
Kathy Yelick: So there are questions about whether you can come up with an incremental algorithm, but

384
01:00:03.900 --> 01:00:13.200
Kathy Yelick: I would say that. I mean, there's there's statistical reasons why you may have thrown out. For example, some of the connections between things when you first ran it that you would want to put back in

385
01:00:13.950 --> 01:00:21.690
Kathy Yelick: If you add some more data that show because it, you're kind of computing a transitive closure. If you're well you're saying, oh, this protein is like this protein.

386
01:00:21.960 --> 01:00:32.670
Kathy Yelick: And it's like something else, but the the first and the third may not be that similar to each other. So it's a little bit hard to know whether you can get an incremental algorithm that will be effective, I think.

387
01:00:32.730 --> 01:00:33.990
Yiqiu Wang: That's very interesting. Thank you.

388
01:00:36.810 --> 01:00:38.940
Julian Shun: Great. Thanks, David.

389
01:00:45.660 --> 01:00:49.590
Julian Shun: David, do you want to unmute yourself if you had a question.

390
01:00:56.790 --> 01:00:58.440
David Reed: You. I hear is terrible. Sorry.

391
01:01:00.420 --> 01:01:06.840
David Reed: The was I was in the middle of raising my hand and I couldn't get back to back to the unmute button.

392
01:01:09.240 --> 01:01:13.110
David Reed: Oh, so as I listened to the algorithm part

393

01:01:14.610 --> 01:01:18.840
David Reed: One thing that struck, which is because I'm not in the field.

394
01:01:19.890 --> 01:01:25.650
David Reed: You're, you're doing fairly small I you know the graph theory or sparse matrix.

395
01:01:27.960 --> 01:01:48.150
David Reed: Operations in particular are very small and and you know there are a lot of them. And ultimately, they may involve moving data in your distributed memory thing and it struck me that from my signal processing background I'm much more current on

396
01:01:49.200 --> 01:02:00.930
David Reed: Third two approaches one old and one much newer for doing some of the same kinds of things that I'm not sure whether they're

397
01:02:02.070 --> 01:02:07.800
David Reed: Hidden in this or whether others. Others are using it. When, when a new approach is called compressed sense

398
01:02:09.600 --> 01:02:21.660
David Reed: And the basic idea of compressed sensing is that you essentially dynamically compress the data using

399
01:02:22.980 --> 01:02:32.100
David Reed: Its own patterns. You know, you build up a library of, you know, when one way of compressing is a library compression based on frequency and stuff like that.

400
01:02:33.570 --> 01:02:37.560
David Reed: And the other older way is transforms

401
01:02:38.640 --> 01:02:52.740
David Reed: If you have, you know, a long SIGNAL. YOU MIGHT DO A for a transformer. Now this space is not, you know, time sequences, but it is a space that's got structure.

402

01:02:53.310 --> 01:03:12.060
David Reed: To it. And you could imagine an algebra of the cavers that allowed you to derive strings from not just from concatenation, but from some kind of difference difference in so you could actually decompose it into spectral kinds of

403
01:03:14.370 --> 01:03:17.670
David Reed: Parts, which then algorithmically.

404
01:03:19.110 --> 01:03:23.460
David Reed: Are much faster because, you know, in the case of 50 it's

405
01:03:24.960 --> 01:03:26.850
David Reed: Linear rather than quadratic

406
01:03:28.350 --> 01:03:42.300
David Reed: So I'm just wondering are there are there people looking at representational issues here or is it all still about peace sort of springs of low level units being compared and matched and so forth.

407
01:03:43.290 --> 01:03:47.250
Kathy Yelick: Yeah, I mean, it's interesting. I haven't thought about. I think the same

408
01:03:48.300 --> 01:03:57.180
Kathy Yelick: Kind of algorithms that you're referring to, but I but but it is the case that fundamentally what you're trying to do, given all of this raw sequencing data.

409
01:03:57.480 --> 01:04:14.070
Kathy Yelick: Is find a very low dimensional basic linear representation that that is, you know, close to the union of all of those reeds, if you will, that all of those reeds map back onto that was some small amount of errors. So the same sort of

410
01:04:15.240 --> 01:04:29.190
Kathy Yelick: Ideas. I have been wondering a little bit about in this space and also whether we should be thinking. I mean, there are these different phases of assembly and there are these different heuristics in here, but I wonder if we shouldn't be thinking about it more like a

411
01:04:30.720 --> 01:04:40.260
Kathy Yelick: You know it, I don't know, an image processing image reconstruction algorithm or something or where you're, you know, this phase is trying to find the the short distance

412
01:04:41.250 --> 01:04:46.920
Kathy Yelick: You know overlaps and this this other part of it's finding the longer distance ones, which I guess would be a little bit like your frequency

413
01:04:47.160 --> 01:05:00.240
Kathy Yelick: Domains in an FFT. So I they aren't an eye. I think what I find is that they're very discreet algorithms right there really based on graphs and the sort of there's an edge and it's either there. It's not there and

414
01:05:01.020 --> 01:05:10.440
Kathy Yelick: And they certainly look at different kinds of edges and these graphs. But I do wonder if there's a more continuous way of looking at this, that would give us better

415
01:05:10.860 --> 01:05:20.610
Kathy Yelick: Better assembly, because as I said, fundamentally, you're just trying to you're trying to find a linear sequence of these characters that map that all the genomes are represented and so

416
01:05:23.130 --> 01:05:27.780
David Reed: Computer science people sort of discovered that discrete

417
01:05:29.010 --> 01:05:33.510
David Reed: You know that when you're trying to do you know multiplication of very large numbers.

418
01:05:34.980 --> 01:05:47.790
David Reed: For a transform on the digits of the number essentially works really well and you know it. And it took a computer scientist, not, not the people

419
01:05:48.510 --> 01:06:03.780

David Reed: You know, trying to do multiplication of numbers to figure that out. So I'm wondering if there's, you know, there's somewhat of a blindness in the genomics community because they're scientists not mathematicians, so be interesting anyway.

420
01:06:04.050 --> 01:06:18.840
Kathy Yelick: Yeah, well, I have to say, Dan Rockstar who did the original which are parallel algorithm. I know we got into this originally just to make it run faster. If I say I like to make things run fast. You know that have clever and hard to realize algorithms and that that one fit the bill, but

421
01:06:19.950 --> 01:06:28.860
Kathy Yelick: But he is a physicist by background. So I think he's pretty perfectly comfortable in a continuous space to but it, but it is true that the the algorithms, I get. And then, and then

422
01:06:29.310 --> 01:06:43.500
Kathy Yelick: On top of the discrete algorithms say graph algorithms is a bunch of heuristics that make me very, you know, comfortable that they're, they're not exactly why this particular set of risks or values are being used is not

423
01:06:44.520 --> 01:06:45.720
Kathy Yelick: It's not always clear

424
01:06:47.370 --> 01:06:50.820
David Reed: That's fascinating. Thank you for your talk. Thanks.

425
01:06:51.960 --> 01:06:54.600
Julian Shun: Great, thanks a lot for the interesting questions.

426
01:06:54.660 --> 01:07:00.780
Julian Shun: Yeah. Um, well, since we're past three o'clock. Already, we'll wrap up the talk.

427
01:07:01.860 --> 01:07:05.370
Julian Shun: And let's thank Kathy again for the very interesting talk.

428
01:07:06.480 --> 01:07:11.160

Julian Shun: We can do another zoom applause and click on the reactions but

429
01:07:12.420 --> 01:07:14.580
Kathy Yelick: Thank you very much. Thanks for having me.

430
01:07:15.000 --> 01:07:15.840
Julian Shun: Yes, thank you.

431
01:07:16.740 --> 01:07:19.650
Kathy Yelick: Nice to have to travel so anyway. Yeah, definitely.

432
01:07:20.520 --> 01:07:21.660
Kathy Yelick: Great. Alright.

433
01:07:22.800 --> 01:07:23.130
Julian Shun: Alright.