Khanh Nguyen

Lab 3:
Compilation : g++ -std=c++11 -o lab3 main.cpp -lpthread
Run: ./lab3 -t <number of threads> -n <degree of poly to test>

In my original design, where I have the working threads return to main their mutation and main does the comparison to check whether or not the mutated coefficients give a better fitness score than the current best fitness score. I realized that in this design, main is the bottleneck where there is only one main checking and pushing the work to the work_queue but many threads pushing the results back to main through the results_queue.

My optimization is to change my mutate function so that the range would be a function of the fitness score. I use the range of (-log(fitness_score), log(fitness_score)) to generate a random double that will be the amount of change of the coefficient. I also added conditions in the mutate function to check for all of the coefficients, each in both direction of the change (either adding the change to coefficient or subtracting from coefficient) to see which of these coefficient and in which direction will the fitness score be lower. If one of the coefficients found returned a better fitness score, than the result will be returned to main thread. This makes sure that all the work returned to main will be of a better fitness score (since we're checking all the possibilities in terms of direction and coefficient). This reduces main's workload of having to check a lot of results and constantly pushing out work that is the same.

I realized that since I have a for loop that checks all the coefficients, it will tend to favor the coefficients that it checks first (since once a coefficient gives a lower score the for loop breaks and the function returns). This is why I added an if condition so that when the fitness score is higher, I would check the larger power coefficients first in a for loop that goes from higher power coefficients to lower power coefficients. This is because a higher power coefficient tend to give a bigger change in fitness score. When the fitness score is smaller, I will check the lower power coefficients first in a for loop that goes from lower power to higher power coefficients.

The optimization works really well for smaller degrees polynomial. As the degree gets bigger, the number of guesses significantly increases. I think the reason why my optimization doesn't work very well with higher degree is because I have a for loop to check every coefficient, which becomes worse as there are more coefficients, which requires more checking of the coefficients. At the same time, it may be because there are more coefficients which cause more variations, making it take a lot longer for the threads to adjust to the find the right coefficient.

It's important to note that these coefficients and numbers are generated randomly. Sometimes, it just happens that the starting coefficients generated is already very close to the points, which require less work and makes it a lot easier for the threads. This can explain the big difference in some of the results above where all the starting

data points are the same (i.e same degree, number of threads etc) but the resulting time is quite different.

I ran this program with 4 threads, with a maximum allowed summed squared distance of the points to be 1.0. The x and y bounds are [-5,5]**.** The optimization works much better on lower degrees, such as 1,2 and 3 but doesn't optimize much for degree 4 and above. Because it is too slow, I have not included the results for degree 4 and above. I did two different sets of data points for each degree polynomial. In total there are 6 entries of data points. For each set, I ran the program three times on the same set of points to compare the results.

Degree	Data points	Result	Time (s)	# guesses	Mean	Max	Min
1	(3.04244,-1.22392)	y = 2.17825 + -1.22392x	0.000389601	2	2.30E-05	2.89E-05	1.70E-05
1	(-2.27708,4.03803)	y = 1.2722 + -0.924837x	0.000909039	13	1.63E-05	3.08E-05	1.20E-05
1		y = 2.30355 + -0.90174x	0.00151234	30	1.35E-05	4.49E-05	1.02E-05
1	(1.65055,-1.22364)	y= -0.876996 + -0.461456x	0.00112247	7	2.58491e-05	3.79E-05	1.8473e-05
1	(-1.90171,-0.473584)	y= -1.14354 + -0.452787x	0.00167621	14	2.19449e-0	3.79E-05	1.72E-05
1		y = -0.99268 + -0.366226x	0.00182917	19	1.94008e-05	4.00E-05	1.44E-05
2	(3.31747,-2.77356)	y = 4.46689 + 0.357331x + -0.805764x^2	0.00324054	49	2.06E-05	4.86E-05	1.44E-05
2	(-2.22928,0.130041)	y = -1.2336 + -0.401767x + -0.014062x^2	0.00296695	39	2.31E-05	4.75E-05	1.44E-05
2	(3.02015,-2.42383)	y = 1.94734 + -0.0753621x + -0.373066x^2	0.00538075	88	2.2395E-05	4.44E-05	1.43E-05
2	(2.88294,3.00841)	y = 0.198086 + 3.00841x + -0.656411x^2	0.00251887	36	1.97E-05	3.88E-05	1.45E-05
2	(4.45176,1.20866)	y = -0.460843 + 2.45474x + -0.465107x^2	0.00329129	71	1.57E-05	4.02E-05	1.0633e-05
2	(0.415735,1.40441)	y = -0.711582 + 2.98541x + -0.571147x^2	0.00443595	99	1.4967e-05	1.05E-05	3.97E-05
3	(0.0729268,2.9544)	y = 2.13152 + 1.80303x + 0.00450724x^2+ -0.102896x^3	24.2355	759104	1.90E-05	0.00012423	1.07E-05
3	(4.9094,-1.1231)	y = 2.13439 + 1.8016x + 0.0044629x^2+ -0.102927x^3	30.8925	963673	1.93E-05	0.00020696	1.06E-05
3	(-1.05765,-0.363712)	y = 3.31968 + 4.40461x + -0.00694134x^2+ -0.218721x^3	22.7304	706539	1.8684E-05	0.00028994	1.06E-05
	(-4.57277,3.91337)						
3	(-3.8441,-3.09367)	y = 1.48429 + -2.26054x + -1.37005x^2+ -0.137425x^3	1.62701	31407	2.46E-05	0.00019152	1.08E+00
3	(1.36682,-4.46711)	y = 0.810263 + -1.33148x + -1.33016x^2+ -0.186022x^3	0.611797	18976	1.87E-05	4.50E-05	1.08E-05
3	(-4.65223,-3.28891)	y = 0.953145 + -1.21987x + -1.39002x^2+ -0.159203x^3	10.4534	357982	2.32138e-05	5.57E-05	1.221e-05
	(-0.245891,1.83793)						

From my results, I can see that for the same set of data points, the results between the three trials don't vary that much, although the number of guesses can vary significantly between one trial and another. This may be due to the random nature of how the coefficients are picked in the beginning and how they're mutated between each thread. At the same time, we can see that while there's not a big difference in the runtime of polynomial of degree 1 and degree 2, from degree 2 to degree 3 there's a quite large jump in the running time as well as the number of guesses. I'm guessing that the runtime vs. degree graph may be more of an exponential graph, which can explain such a big difference. This may explain why for degree 4, the running time is really high. The mean, maximum and minimum between the threads stay roughly the same regardless of the degree of the polynomial, which makes sense because within each thread, the amount of work in each thread is the roughly the same and doesn't vary much between them. It's the total amount of work for all the thread that varies a lot depending on the degree.

Khanh Nguyen