Filename	Input Values	Sum of Values	ArraySum.adb
5numbers.txt	Five	150	2194
1000numbers.txt	One thousand	50104	14995
10000numbers.txt	Ten thousand	506090	143880
100000numbers.txt	One Hundred Thousand	5059169	1086030
1000000numbers.txt	One Million	50473230	3054085
10000000numbers.txt	Ten Million	505053538	6139674
100000000numbers.txt	One Hundred Million	5050391777	35505089

Filename	Input Values	Sum of Values	array_sum.rb (sec)
5numbers.txt	Five	150	0.000003146
1000numbers.txt	One thousand	50104	0.00003374
10000numbers.txt	Ten thousand	506090	0.000322578
100000numbers.txt	One Hundred Thousand	5059169	0.003242293
1000000numbers.txt	One Million	50473230	0.032212856
10000000numbers.txt	Ten Million	505053538	0.321191712
100000000numbers.txt	One Hundred Million	5050391777	3.235100756

Filename	Input Values	Sum of Values	array_sum.adb (sec)
5numbers.txt	Five	150	0.000002
1000numbers.txt	One thousand	50104	0.000023
10000numbers.txt	Ten thousand	506090	0.000043
100000numbers.txt	One Hundred Thousand	5059169	0.000419
1000000numbers.txt	One Million	50473230	0.004107
10000000numbers.txt	Ten Million	505053538	0.041058
100000000numbers.txt	One Hundred Million	5050391777	0.410342

Java

1	2	3	4	5
340481	554688	687033	795017	1084598
342277	607225	644310	757148	828214
521227	664115	727532	752461	853115
1450683	1446638	1639596	1642074	1605645
3555012	5062700	6692376	8098149	10715564
7179360	7035085	6749105	6243595	8566881
38070191	28326794	24609043	24110977	24609043

Java 1. One thread compared to ArraySum is slower because the program has to run the extra code required to thread the array, but only uses one thread, which is essentially the same thing as running it without any threading.

Java 2. The smaller files are being processed significantly slower than ArraySum. The larger the text file becomes the more efficient threading becomes because the number of summations that is being divided between threads is increasing, but the threading still slows the initial process as seen with a single thread compared with ArraySum. More threads with the smaller files seems to decrease the speed because the program is spending more time on making threads. The file containing one hundred million numbers is much faster than processing the file without threading because it is so large.

Java 3. When the file is small the program prioritizes more time to making threads than is worthwhile and causes the program to run slower.

Java 4. When I run the summation of one million numbers without the JIT compiler I receive an output time of 10792712 rather than 3555012, which is three times slower than with the compiler.

Ruby

1	2	3	4	5
0.000004567	0.00011181	0.000108389	0.00042218	0.00033968
0.000046661	0.00010461	0.000144669	0.0004551	0.0003745
0.000444474	0.00057245	0.000621182	0.00092814	0.00100761
0.0042118	0.00434864	0.004395184	0.00465604	0.00478428
0.041056885	0.04221481	0.042759132	0.04265676	0.04357376
0.410177097	0.41748189	0.415226195	0.4244597	0.41642868
4.13121752	4.158740005	4.112315653	4.224064234	4.247364223

Ruby 1. array_sum is consistently faster than threaded_array_sum with a single thread by a small amount

Ruby 2. increasing the number of threads increases the speed of processing the files

Ruby 3. (C++ was not in this lab, so the answer is there is no way of comparing) Comparing it to Java, however, it appears that ruby runs faster, but I am unclear about Java's time unit (appears to be nanoseconds), and I would have guessed that Java would run faster because ruby is so loosely typed and puts more work on the compiler, but maybe the method we used for multi-threading is significantly faster than the method available to Java.

Ada

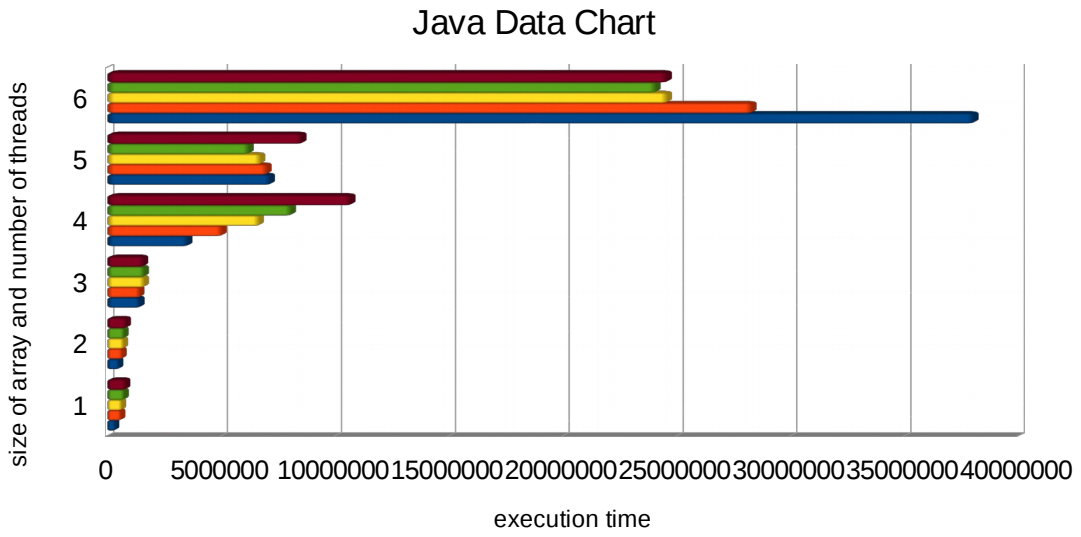
1	2	3	4	5
0.000005	0.00034	0.000707	0.00051	0.001148
0.000019	0.000297	0.0007	0.000745	0.000678
0.000174	0.000535	0.000407	0.000593	0.000534
0.00043	0.00047	0.000542	0.000569	0.000734
0.00413	0.002485	0.001889	0.001691	0.002109
0.041064	0.022271	0.015429	0.015429	0.016598
0.410394	0.207346	0.150615	0.114384	0.161359

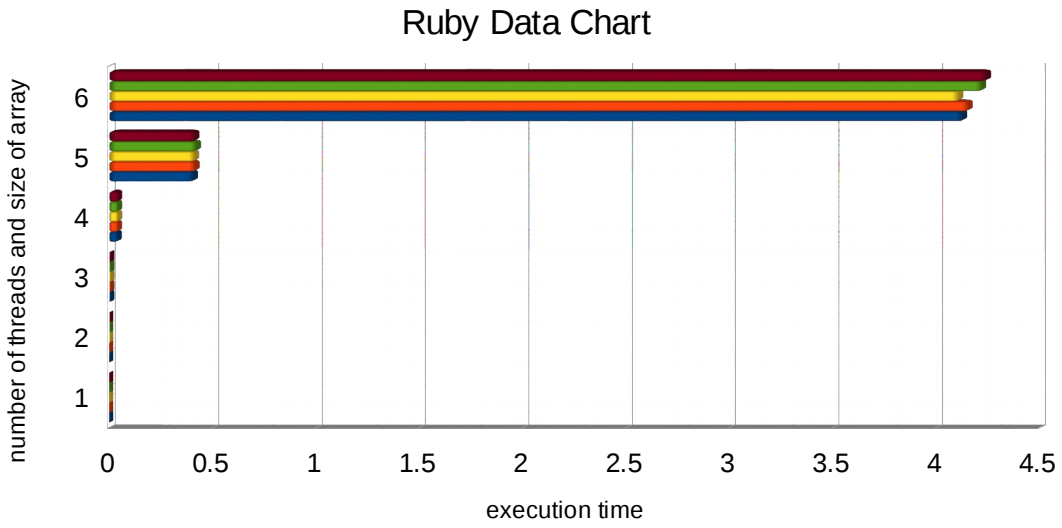
Ada 1. The array_sum calculations were faster because threaded_array_sum had to do the extra steps of creating a thread.

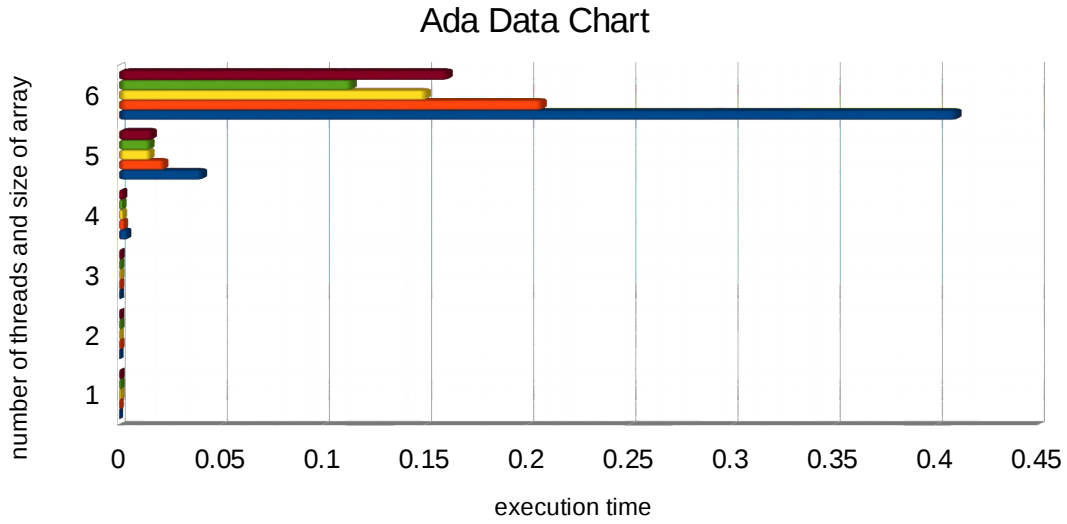
Ada 2. Increasing the number of threads typically increased the speed of the calculations. Sometimes the number of threads slows down the process. This is probably because the computer has to keep track of so many processes at once.

Ada 3. It is not beneficial until about the one million mark with both 2 and 4 threads.

Ada 4. Ada appears to be significantly slower than Java probably because Java uses a better interpertter.


Java Data Chart
size of array and number of threads
6
5
4
3
2
1
0
5000000
10000000
15000000
20000000
25000000
30000000
35000000
40000000
execution time


Ruby Data Chart
number of threads and size of array
6
5
4
3
2
1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
execution time


Ada Data Chart
number of threads and size of array
6
5
4
3
2
1
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
execution time