Mikhail Andreev

EC527 Assignment 2

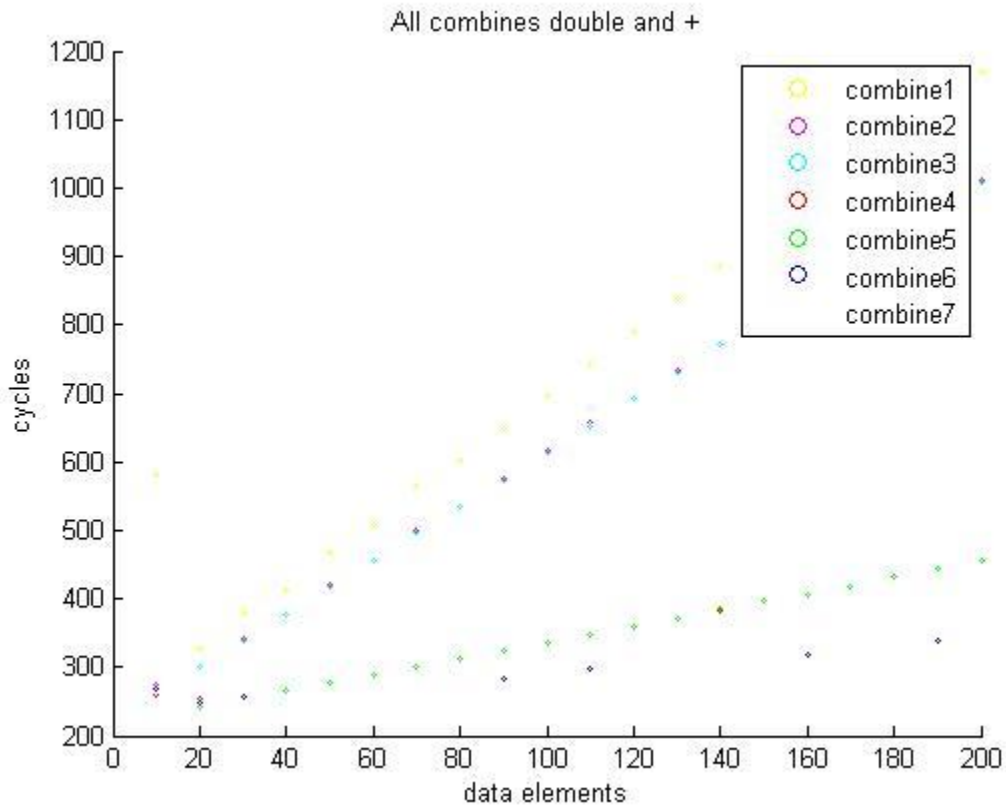
Using hpcl-18 (2.53 GHz)

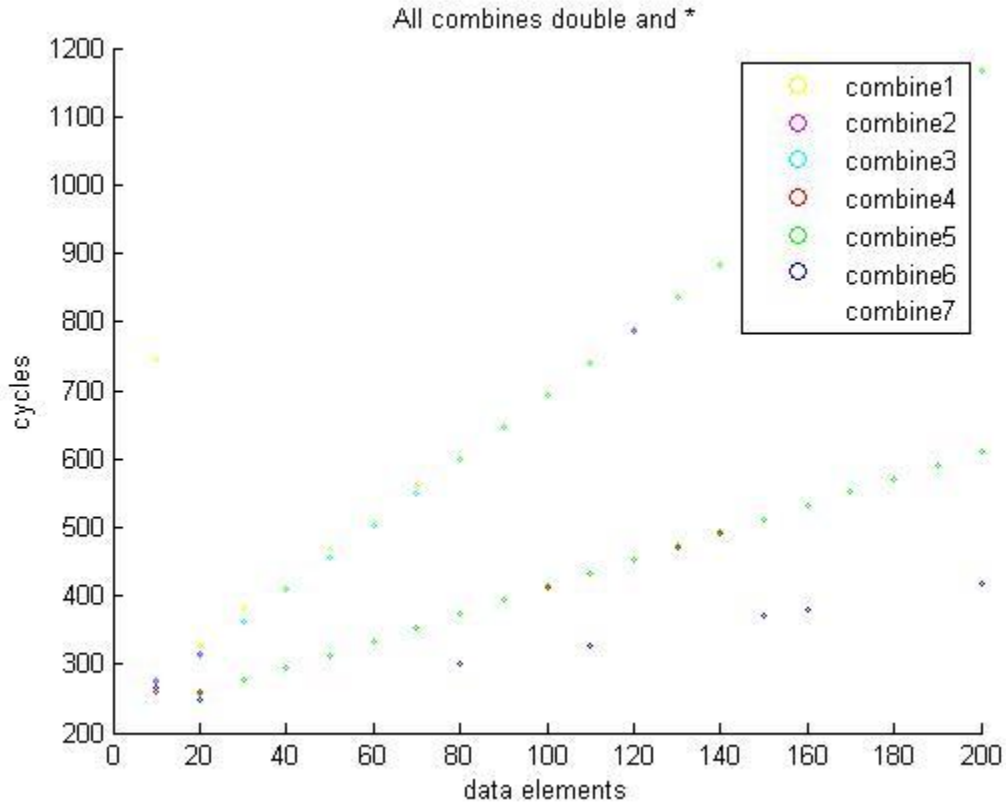
Question 1: Experiment with basic optimization methods

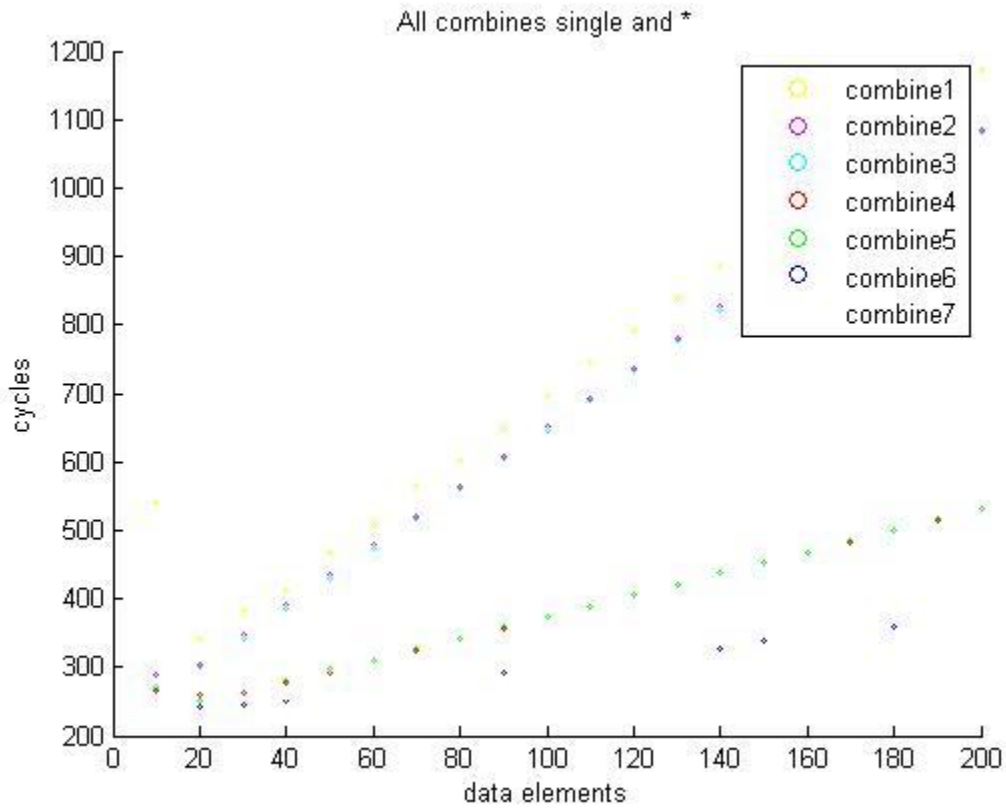
Part a:

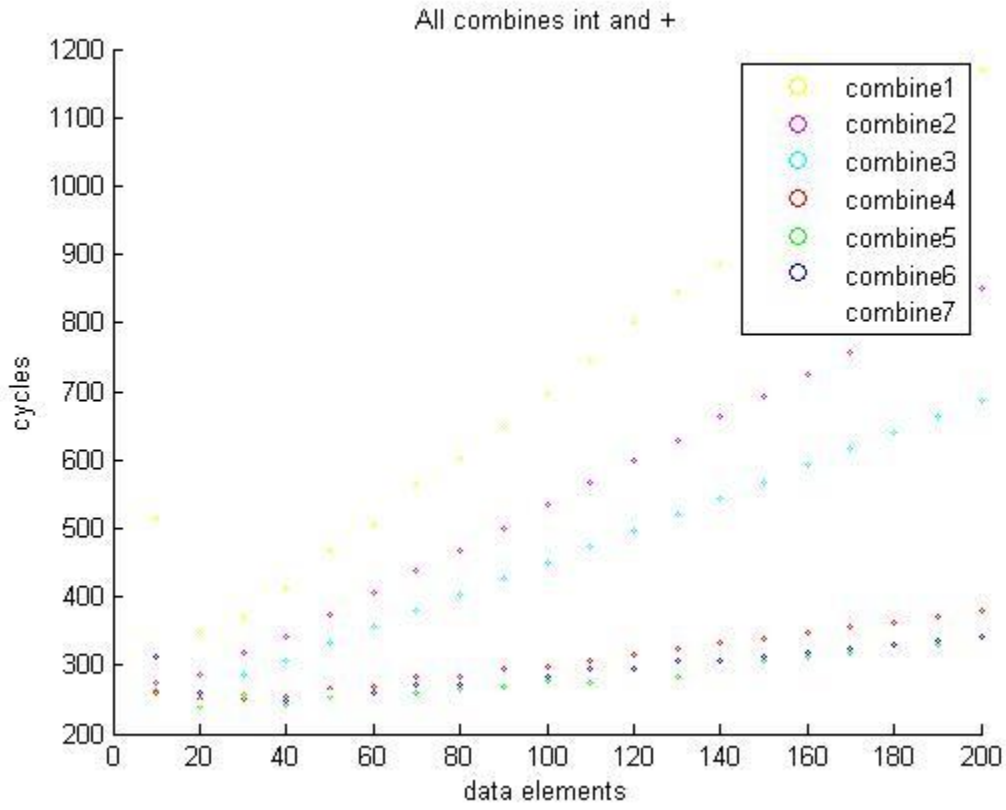
	int +	int *	single *	double +	double *
combine1	5.845	5.87	5.855	5.845	5.845
combine2	4.255	4.275	5.425	5.055	5.83
combine3	3.43	4.245	5.405	5.04	5.83
combine4	1.9	2.28	2.66	2.275	3.055
combine5	1.69	1.735	2.66	2.28	3.055
combine6	1.705	1.73	1.885	1.715	2.085
combine7	1.7	1.695	1.885	1.715	2.08

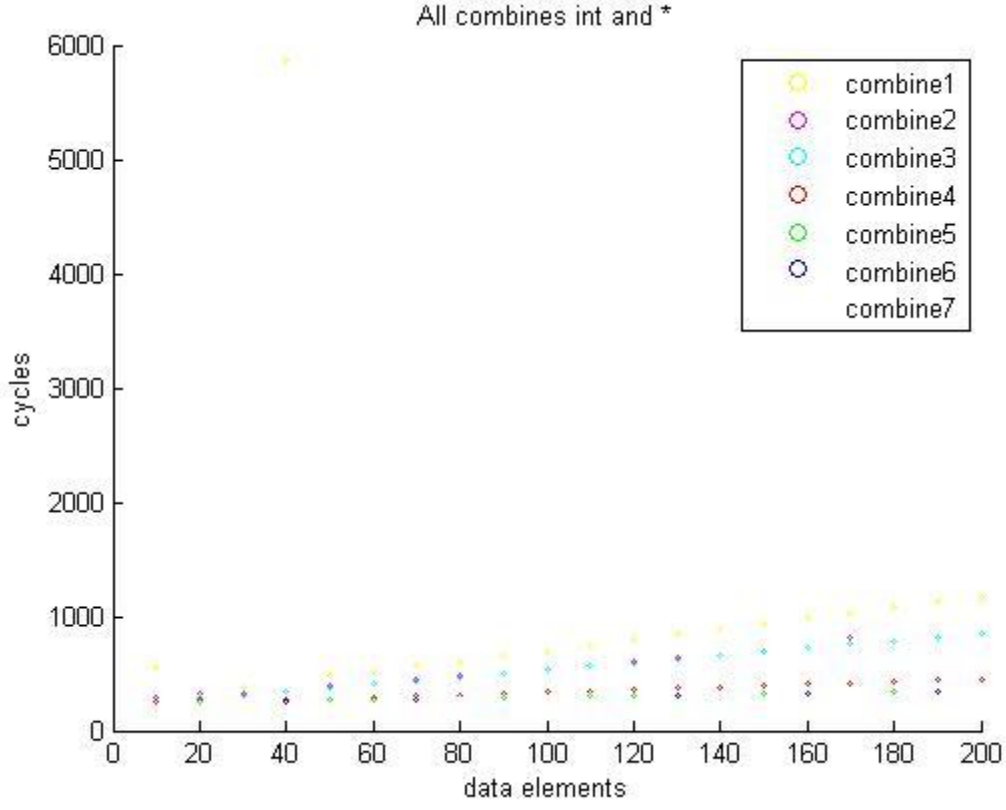
Using the values for combine7, we see that they are slightly larger than those presented in Bryant and O'Halloron. The most likely reason for this is that the vector sizes are small, meaning that function overhead plays a larger role. For double multiplication, this value is smaller than that presented in B&O. The reasoning behind this is that the chip design has changed slightly to include more floating-arithmetic units, allowing for greater throughput of the multiplication.


All combines double and +
combine1
combine2
combine3
combine4
combine5
combine6
combine7
cycles
data elements
1200
1100
1000
900
800
700
600
500
400
300
200
0
20
40
60
80
100
120
140
160
180
200

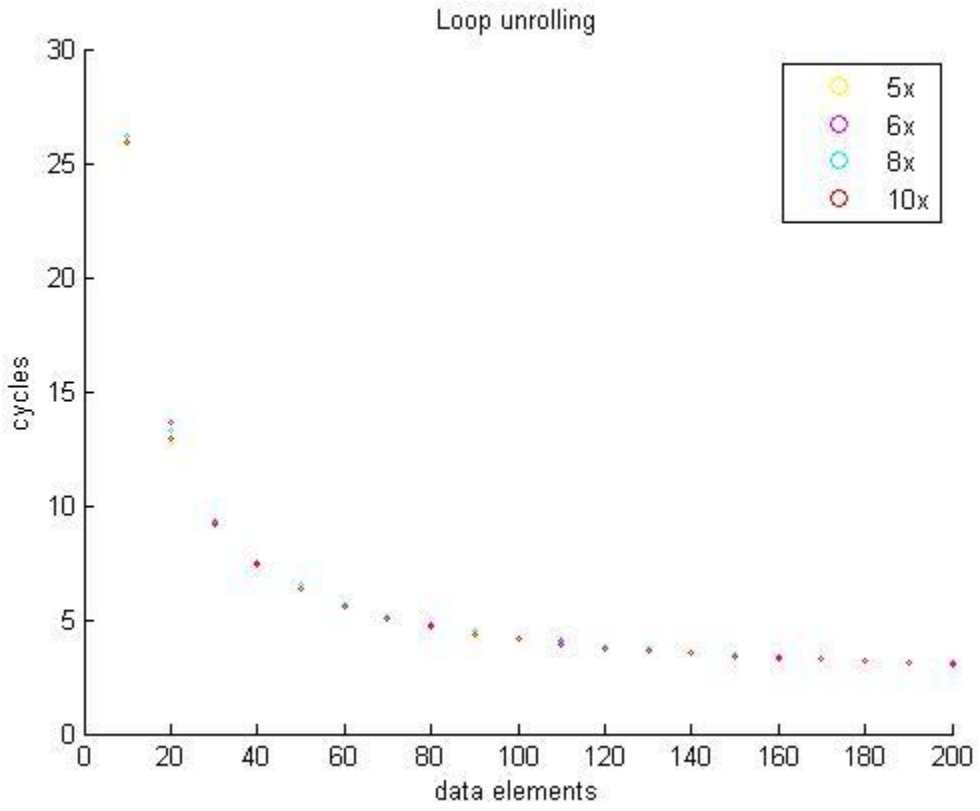

All combines double and *
combine1
combine2
combine3
combine4
combine5
combine6
combine7
cycles
data elements
1200
1100
1000
900
800
700
600
500
400
300
200
0
20
40
60
80
100
120
140
160
180
200


All combines single and *
combine1
combine2
combine3
combine4
combine5
combine6
combine7
cycles
data elements
1200
1100
1000
900
800
700
600
500
400
300
200
0
20
40
60
80
100
120
140
160
180
200


All combines int and +
combine1
combine2
combine3
combine4
combine5
combine6
combine7
cycles
data elements
0
20
40
60
80
100
120
140
160
180
200
200
300
400
500
600
700
800
900
1000
1100
1200

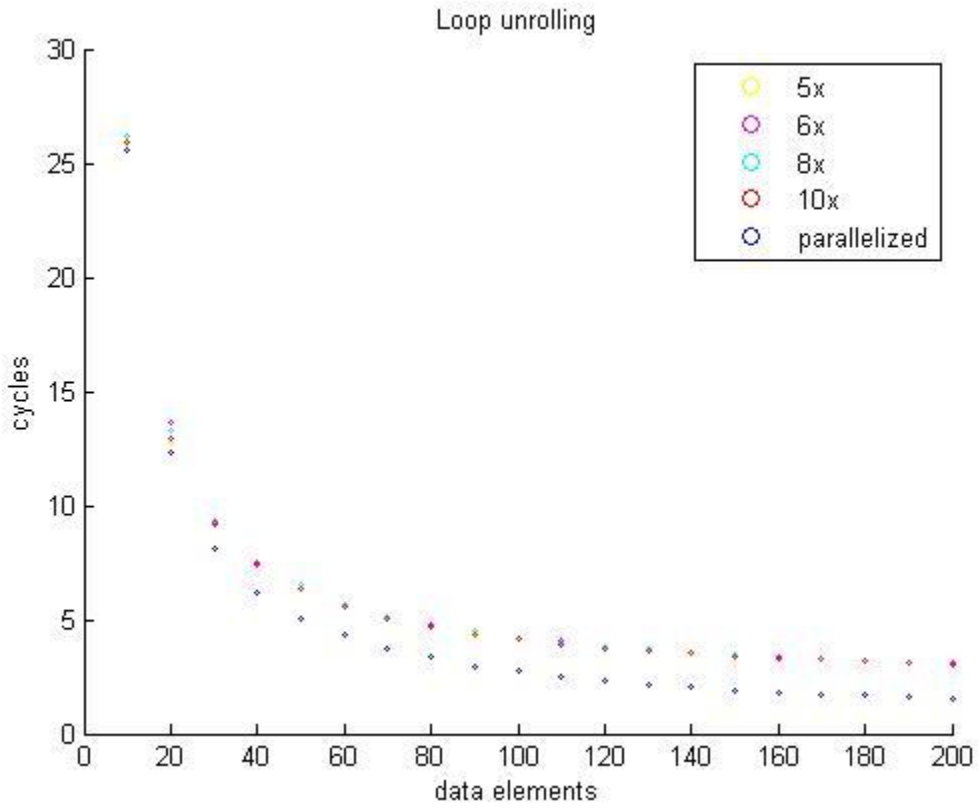

All combines int and *
combine1
combine2
combine3
combine4
combine5
combine6
combine7
cycles
data elements
0
20
40
60
80
100
120
140
160
180
200
0
1000
2000
3000
4000
5000
6000

Part b:



There is no noticeable decrease in performance after loop unrolling x6. The reason that could happen could be because the code becomes too large to fit into the cache, so in order for the processor to run the code, it has to fetch those instructions from memory. From here we can see that this did not occur in this case because the code was small enough to fit into the cache.

Part c:

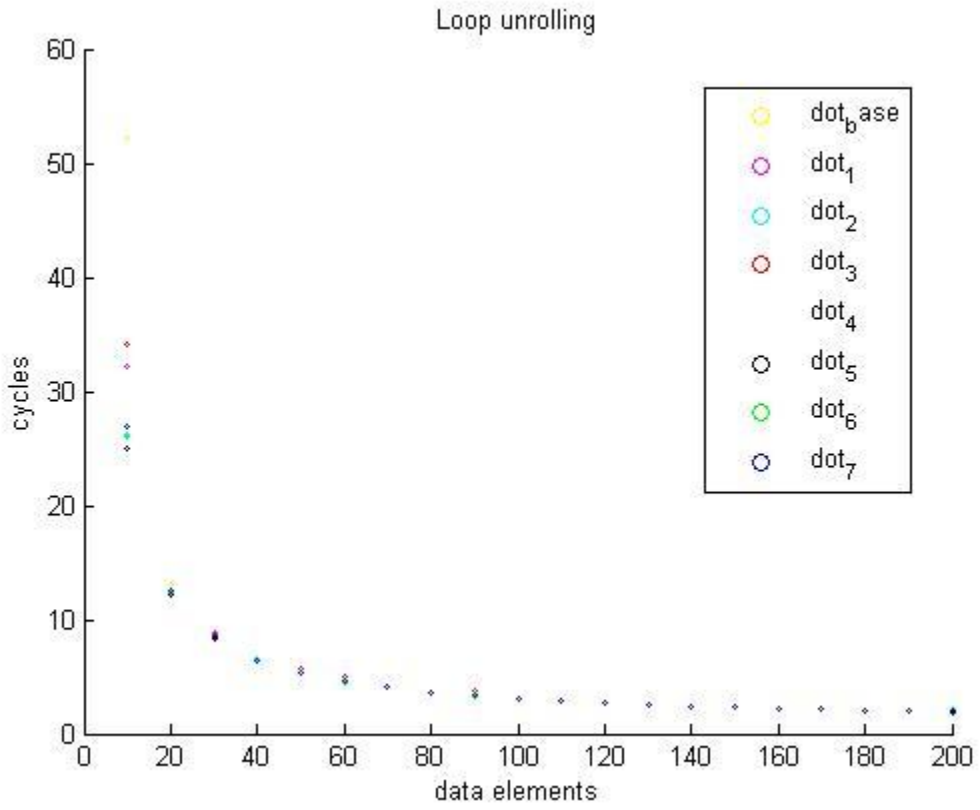


By using parallelization, we can decrease the CPE by half over the best loop unrolling result.

Question 2: Apply basic methods to dot product

Base dot product	2.29
loop unrolling x2	2.3
loop unrolling x4	2.28
loop unrolling x8	2.285
loop unrolling x10	2.285
parallelization x2	1.945
parallelization x5	1.925
reference association	1.915

The code demonstrates progressive optimizations done to the dot product. The initial base product performs a basic dot product. The first four optimizations perform a series of loop unrolling optimizations. The next two optimizations attempt to parallelize the code with multiple accumulators. After the final optimization uses reference associativity to increase performance.



Q3: Force and evaluate conditional branches

Four data sets were generated:

I: one vector is consistently larger than the other, so prediction is very simple

II: One vector is larger than the other, but then switches to being smaller halfway through

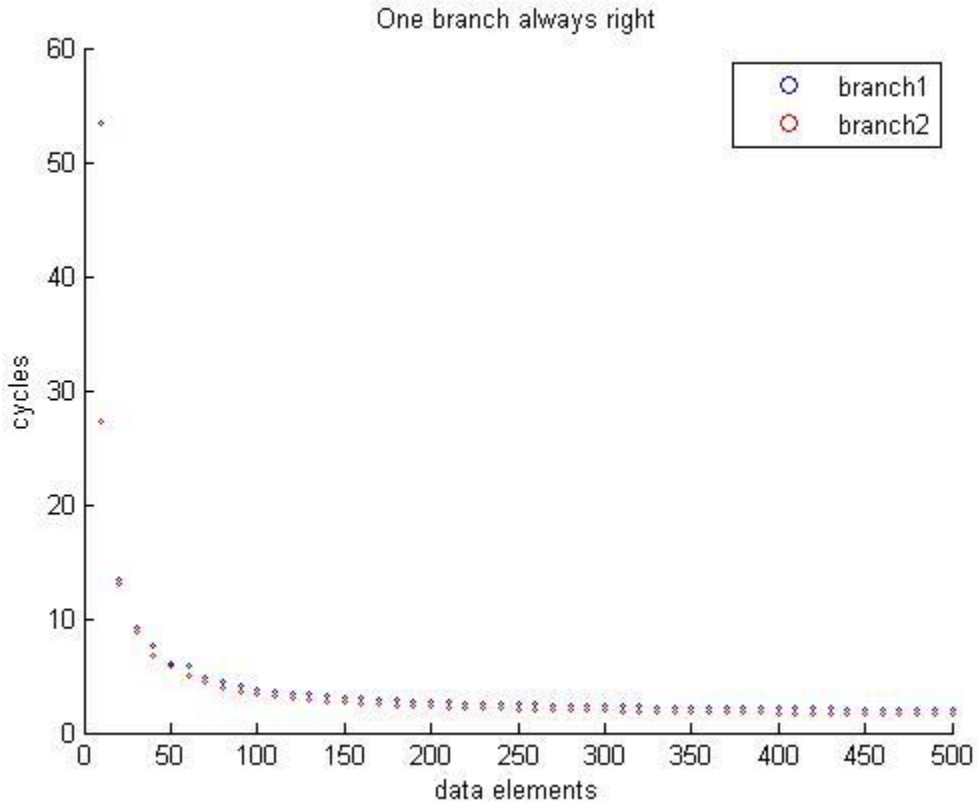
III: Vectors constantly interchange which is larger

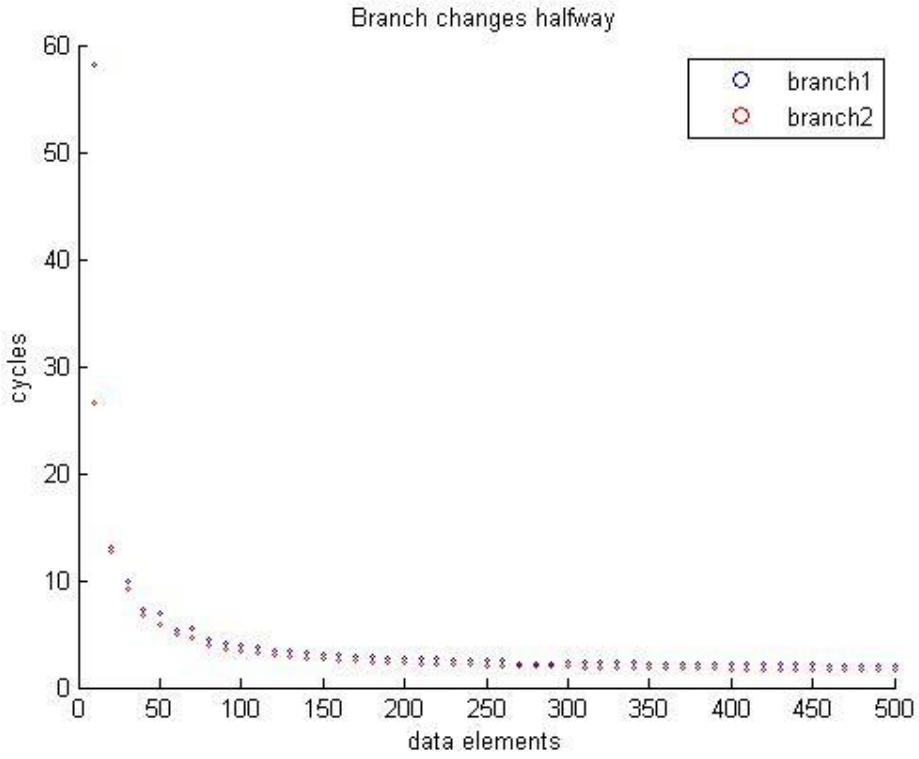
IV: Vector values are random in the range of 0 to 100

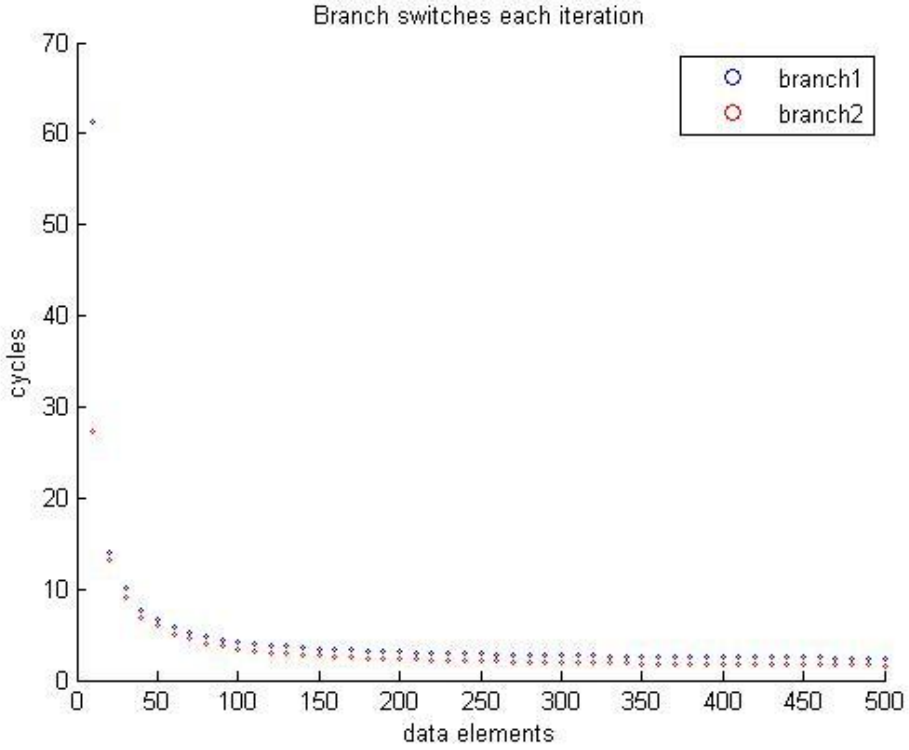
One choice	2.036	1.634
split	2.064	1.634
Up/Down	2.424	1.634
Random	7.36	1.65

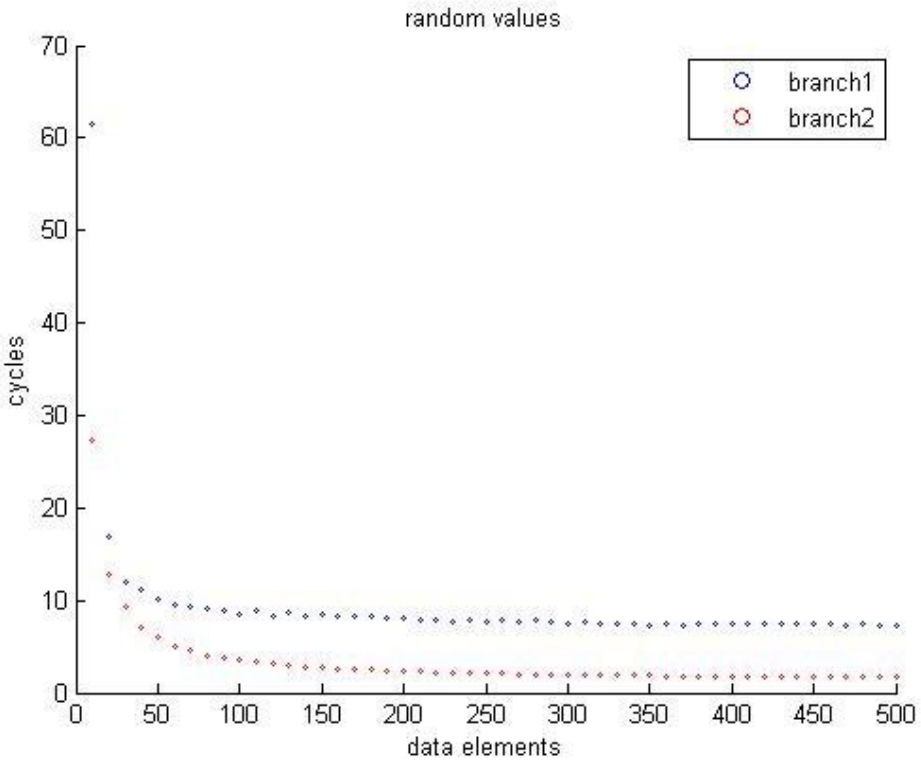
The one choice data set is very easy to predict, so the CPE is not much worse with the branch prediction. When introducing the split, it does cause a bit of a cycle delay, but not very noticible. When introducing the up/down variation where the values oscillate between the arrays, we see a larger increase in cycle

delay. However, even there the branch predictor is able to pick up a pattern. When using the random values, there is no pattern present, so the branch prediction is consistently wrong.

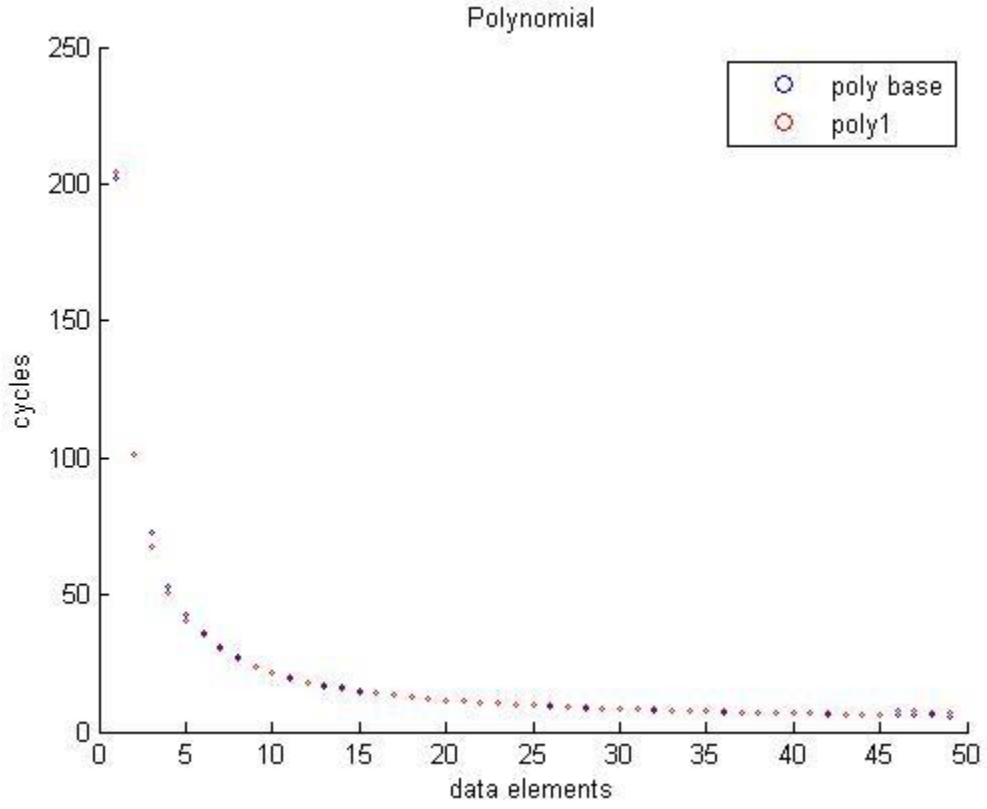



Branch changes halfway
branch1
branch2
cycles
data elements
0
10
20
30
40
50
60
0
50
100
150
200
250
300
350
400
450
500


Branch switches each iteration
branch1
branch2
cycles
data elements
0
10
20
30
40
50
60
70
0
50
100
150
200
250
300
350
400
450
500


random values
branch1
branch2
cycles
data elements
0
10
20
30
40
50
60
70
0
50
100
150
200
250
300
350
400
450
500

Q4: Optimizations on a slightly more complex application



With loop unrolling we don't get any noticeable increase in performance.