Mikhail Andreev

EC527 Assignment 2

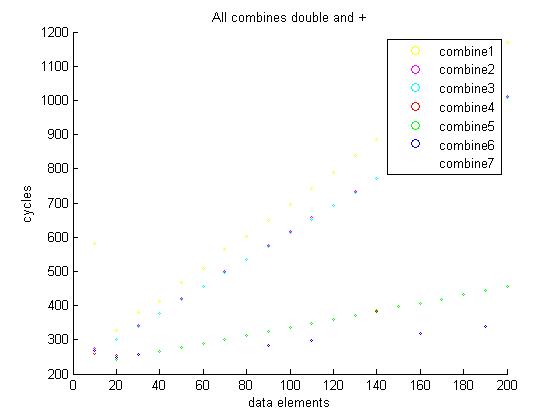
Using hpcl-18 (2.53 GHz)

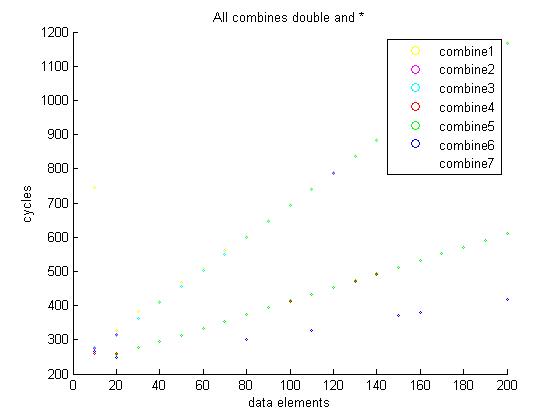
**Question 1: Experiment with basic optimization methods**

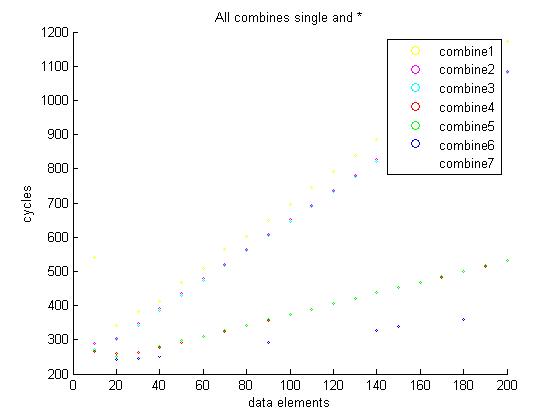
Part a:

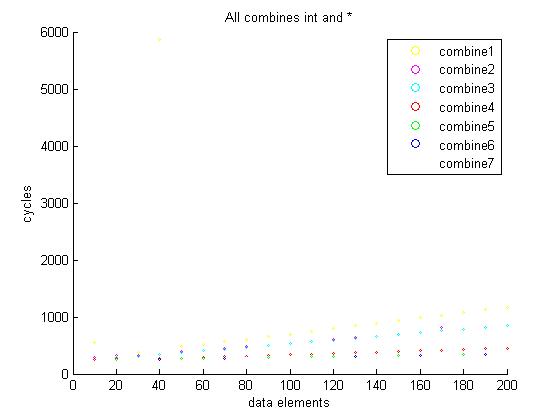
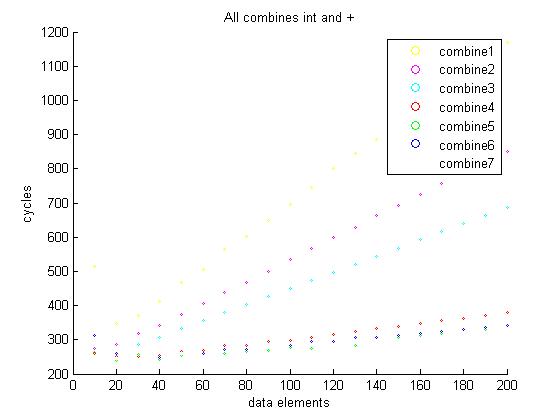
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | int + | int \* | single \* | double + | double \* |
| combine1 | 5.845 | 5.87 | 5.855 | 5.845 | 5.845 |
| combine2 | 4.255 | 4.275 | 5.425 | 5.055 | 5.83 |
| combine3 | 3.43 | 4.245 | 5.405 | 5.04 | 5.83 |
| combine4 | 1.9 | 2.28 | 2.66 | 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.

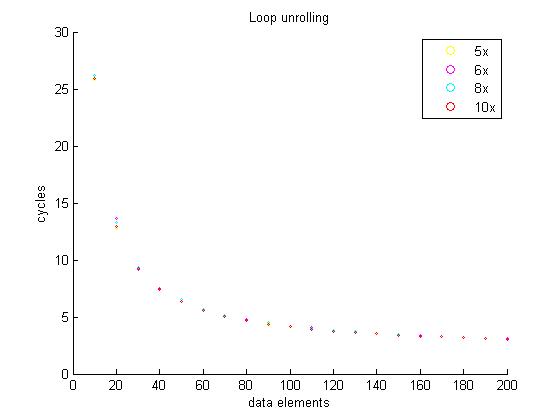






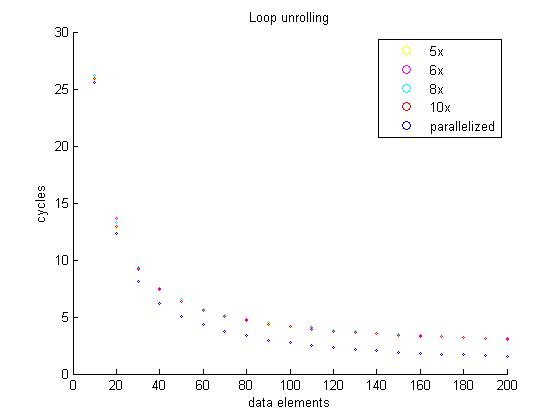


Part b:



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**