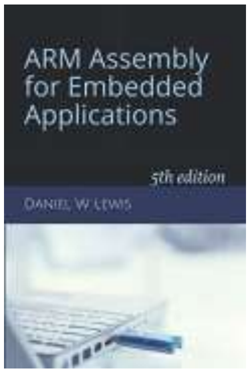


Programming Lab 12A

Scaling Data with SIMD

Click to download Lab12A-Main.c

Topics: SIMD processing, finding minimum and maximum values of an array.

Prerequisite Reading: Chapters 1-12
Revised: January 1, 2021

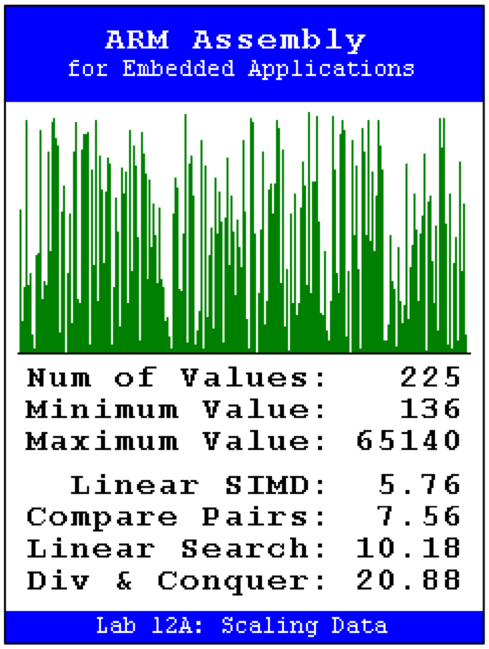
Background: Plotting a set of values on a bar graph requires determining the minimum and maximum values (i.e., the *range*) so that the data can be scaled to fit the screen coordinates of the graph. There are three fairly well-known algorithms for finding the minimum and maximum: linear (sequential) search, comparing adjacent pairs, and divide and conquer. All three solutions are $O(n)$, but differ in their execution time due to differences in the number of comparisons they require.

	# compares:	
Algorithm	***Best case***	***Worst case***
Sequential search	$2n-1$	$n-1$
Comparing pairs	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$
Divide and conquer	$\frac{3n}{2}-2$	$\frac{3n}{2}-1$

Assignment: The main program includes all three of these algorithms and a fourth named `LinearSIMD` that is a modified linear search using SIMD instructions that makes multiple comparisons simultaneously. I.e., you can compile and run the program without writing any assembly. However, your task is to create an equivalent replacement for `LinearSIMD` in assembly using the C version to guide your implementation. The original C version has been defined as "weak", so that the linker will automatically replace it in the executable image by the one you create in assembly; you do not need to remove the C version.

The first parameter of function `LinearSIMD` is the starting address of a word-aligned array of 16-bit unsigned integers; the second is the number of items in the array. The function returns a 32-bit word in register `R0`, with the minimum value in the least-significant 16 bits and the maximum in the most-significant 16-bits.

The main program creates a random array of integers, measures the execution time of all four algorithms, and then uses the minimum and maximum values to scale the array so that it may be plotted on the display. This process repeats indefinitely five times per second unless an error occurs. If your code is correct, the display will look like the one shown. The numbers in the last four rows indicate performance as measured by the total number of clock cycles of each algorithm divided by the number of items in the array[1]. If your code contains an error, its results will be displayed as white text on a red background and testing will pause; pressing the blue pushbutton will cause testing to resume. *To receive full credit, your SIMD assembly version must be faster than all three of the other algorithms.*



```
ARM Assembly
for Embedded Applications

Num of Values:    225
Minimum Value:    136
Maximum Value:  65140

  Linear SIMD:   5.76
Compare Pairs:   7.56
Linear Search:  10.18
Div & Conquer:  20.88
Lab 12A: Scaling Data
```

[1] ***Note:*** The main program contains both a recursive and an iterative implementation of the divide and conquer algorithm. The recursive version is not used, but provides a clearer representation of the algorithm; the iterative version is used by the program for better run-time performance. Although the divide and conquer algorithm and the comparing pairs algorithm both use about the same number of comparisons, the additional overhead of stacking and unstacking parameters causes the divide and conquer implementation to be significantly slower.