

Assignment 5

Liam Roeth

CSC 153

I defined three versions of the function `dot_product(a, b, size)`, each implementing a different algorithm:

<code>dot_product_nosimd()</code>	No AVX optimization. Multiply floating point numbers via regular <code>*</code> , then add via <code>+</code> .
<code>dot_product_simd_addmul()</code>	Multiply vectors with <code>_m256_mul_ps</code> , then add vectors of products directly via <code>_m256_add_ps</code> .
<code>dot_product_simd_dp()</code>	Dot-product half of each pair of vectors via <code>_m256_dp_ps</code> ; then merge four such resulting pairs of dot products via <code>_m256_blend_ps</code> ; then add blended vectors via <code>_m256_add_ps</code> .

Then, I made each selectable via commandline options. I then ran, using array size 1000000, a bash test script that compiled the times of 100 runs of each.

Function	Time (μ sec)
<code>dot_product_nosimd()</code>	3530.55685
<code>dot_product_simd_addmul()</code>	1618.14716
<code>dot_product_simd_dp()</code>	1314.45971

Table 1: Mean execution times

Results show a massive speed increase (118.185%) due to vector optimization, and a small, but significant increase (23.1036%) from the direct multiplication and addition to the native dot product function. Results were not significantly variable with any algorithm.

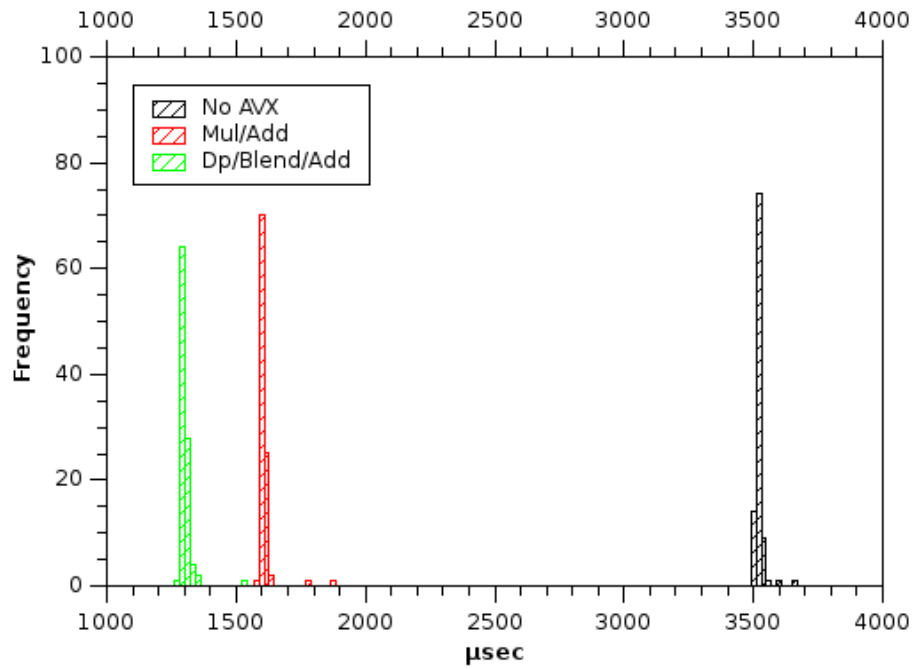


Figure 1: Execution time histogram

```
#!/bin/bash
n=100
exe="$1"
arg="$2"
while [[ n -gt 0 ]]; do
    "$exe" -n "$arg"
    echo -n ","
    "$exe" -a "$arg"
    echo -n ","
    "$exe" -d "$arg"
    echo ""
    n=$(( n-1 ))
done
```

Figure 2: Test script