1. One of the primary advantages of vectorized operations is that they allow for parallel use of operands upon multiple data in a manner that is predictable and easy to implement. Unlike multiple thread or other forms of parallelism, vectorized operations need not wait for other tasks complete operation, worry about synchronicity or spin lock, and involve minimal overhead. However, such parallelism is highly hardware dependent, with the number of items per register and number of vector-enable registers dependent upon the processor and instruction set used by the processor.
2. The use of *unsigned char* failed to provide any difference in performance, with the programing taking approximately 2.0e-6 seconds to find the complement a string of one-hundred and forty-six characters. This was the same runtime as the *unsigned long* implementation. The root of this result likely lies in more individual blocks requiring computation. While it is faster to pack less bits into an *unsigned char*, less bits are available per vectorized computation, resulting in more blocks needing to be computed and thus offsetting any potential speedup. However, upon implementing the *unsigned long long* version, a speedup of two-times was observed, with the programing finding the complement of same string of length one-hundred forty-six in 1.0e-6 seconds. Unlike the char implementation, the *unsigned long long* implementation takes full advantage of the number of bits available per register (per vector), packing in ans many characters as possible into a single operation. As a result, less blocks require computation.

Likewise, for a string of size 2191 characters, the *unsigned long long* implementation achieved two-fold speedup, producing the correct complement within approximately 3.3e-5 seconds. Clearly, by using vector parallelism to its fullest (packing as many bits into a vector as can be fit in a single register), the impact of this form of parallelism can be maximized.