We think about approaching an optimization problem of converting serial code to SIMD-ized code as follows:

a. What are your SIMD intrinsic sizes? (\texttt{_m128i} is 128-bits)

b. What are you iterating over?

c. How many parallel operations can you perform given a) and b) per iteration of a loop?

d. What is your base case?

e. Perform operations in loop

f. Store your SIMD intrinsics back into accessible primitives (ints, floats, etc)

g. Tail case / return

Applying this logic to the problem:

a. 128-bits (we are working with \texttt{_m128i}'s)

b. ints (4 bytes = 32 bits)

c. $128/32 = 4 \Rightarrow$ we can process 4 ints in parallel at a time

d. We word a product so we want a number $x$ such that $\forall a \in \mathbb{N}$, $xa = ax = x \Rightarrow a = 1$. So we start with a 1 for our base case.

e. Multiply 4 ints at a time

With this information, we can explain everything in the main loop. Our base case is 1, so we load our 128-bit intrinsic with 4 1's in 32-bit slots.

So initially, our intrinsic looks like this

\begin{verbatim}
128  96  64  32  1
0x1  0x1  0x1  0x1
\end{verbatim}

We then process 4 ints at a time.

Thus we need to bring $n$ down to its closest multiple of 4. Mathematically, we want $n - n \% 4$. This is exactly equivalent to floor dividing $n$ by 4 and then re-multiplying back $(n/4 \times 4)$; we also increment by 4 ints each time as well ($i+ = 4$).

Inside the loop, all we are doing is multiplying our current product (prod-v) by the next 4 ints \texttt{(_mm_loadu_si128(_m128i*)(a+i))}. Note, the parenthesis are important because \texttt{(_m128i*) a + i == ((_m128+i)a)+i} which means you are not advancing your array by ints but rather 128-bit intrinsics.
Afterwards, we store things back from our 128-bit intrinsic into something we can access (i.e. an int array). Now we have four separate products in 4 elements of result.

Since we truncated our original loop, we need to consider the last \( n \% 4 \) elements; this is our tail case. We can’t use the SIMD here since there are less than 4 elements. We loop from \( n/4 \times 4 \) to \( n \) doing serial multiplication.

At the end, we combine our few disjoint products into a final, total product.

Note: for SIMD-izing functions, it must be that the operation that we perform is generally associative and commutative.