我们有两个任务：a) 优化SM4的软件实现，b) 实现并优化SM4-GCM模式。

首先，我们需要一个基本的SM4实现，然后逐步进行优化。

优化手段包括：T-table、使用AESNI指令集、以及使用最新的指令集（如GFNI和AVX512中的旋转指令VPROLD等）。

步骤：

1. 实现基本的SM4算法（包括密钥扩展和加密/解密函数）。

2. 使用查表法（T-table）优化。

3. 使用AESNI指令优化S盒变换。

4. 使用GFNI指令进一步优化（如果可用）。

5. 使用AVX512等指令集进行并行化处理（例如一次处理多个块）。

6. 实现SM4-GCM模式，并利用上述优化进行加速。

我们先从基本的SM4实现开始。

SM4算法描述：

- 加密过程：32轮迭代，每轮使用一个轮密钥。

- 每一轮操作：

X\_{i+4} = F(X\_i, X\_{i+1}, X\_{i+2}, X\_{i+3}, rk\_i) = X\_i xor T(X\_{i+1} xor X\_{i+2} xor X\_{i+3} xor rk\_i)

其中T函数由非线性变换τ（S盒）和线性变换L组成：T(.) = L(τ(.))

S盒是一个固定的8比特输入8比特输出的置换表。

L变换：B = L(A) = A xor (A<<<2) xor (A<<<10) xor (A<<<18) xor (A<<<24)

解密过程与加密相同，只是轮密钥逆序使用。

密钥扩展：

首先将密钥MK=(MK0, MK1, MK2, MK3)与系统参数FK=(FK0, FK1, FK2, FK3)异或得到K0~K3。

然后对于i=0~31：

rk\_i = K\_{i+4} = K\_i xor T'(K\_{i+1} xor K\_{i+2} xor K\_{i+3} xor CK\_i)

其中T'变换与T类似，但线性变换L'不同：L'(A)=A xor (A<<<13) xor (A<<<23)

T'中的S盒与T中的相同。

系统参数FK和固定参数CK都是给定的常量。

基本实现步骤：

1. 定义S盒（256字节）、FK（4个32位字）、CK（32个32位字）。

2. 实现线性变换L和L'，以及T和T'函数。

3. 密钥扩展：输入128位密钥，输出32个轮密钥（每个32位）。

4. 加密函数：将128位明文分为4个字，然后进行32轮迭代，最后输出4个字的密文。

接下来，我们逐步优化。

优化1：T-table

在AES中，T-table将一轮中的多个步骤合并成查表操作。类似地，我们可以为SM4构造T-table。

但注意SM4的一轮操作是：

X\_{i+4} = X\_i xor T(X\_{i+1} xor X\_{i+2} xor X\_{i+3} xor rk\_i)

其中T函数包括S盒和L变换。我们可以将T函数对32位输入的处理分成4个S盒（每个字节一个S盒）然后进行L变换。

然而，由于L变换是线性且跨越字节的，我们无法直接构造4个独立的表（像AES那样）。但是，我们可以将T函数拆分成4个查表（每个表处理输入的一个字节）然后通过异或合并。具体如下：

设Z = X\_{i+1} xor X\_{i+2} xor X\_{i+3} xor rk\_i，将Z分成4个字节：Z = (z0, z1, z2, z3)

则T(Z) = L(τ(Z)) = L( Sbox(z0), Sbox(z1), Sbox(z2), Sbox(z3) )

我们可以将L变换拆分成4个部分，每个部分对应一个字节的贡献。但是L变换是线性的，因此可以表示为：

T(Z) = [Sbox(z0) \* 某个线性变换] ⊕ [Sbox(z1) \* 某个线性变换] ⊕ ...

具体地，设Sbox(z0)为a0, Sbox(z1)为a1, Sbox(z2)为a2, Sbox(z3)为a3，则：

T(Z) = L( a0<<24 | a1<<16 | a2<<8 | a3 )

= (a0<<24 | a1<<16 | a2<<8 | a3)

⊕ (a0<<24 | a1<<16 | a2<<8 | a3)<<<2

⊕ ... (其他移位)

我们无法像AES那样将整个T函数拆分成4个独立的表（因为移位操作会跨字节），但是我们可以预先计算一个32位输入到32位输出的T-table（即整个T函数），但这样表的大小为2^32\*4字节，显然不可行。

因此，我们采用另一种方法：将T函数拆分成4个8位输入32位输出的表，每个表对应输入的一个字节，然后通过异或合并。具体构造方法如下：

将T函数分解为：

T(Z) = T0(z0) xor T1(z1) xor T2(z2) xor T3(z3)

其中，T0(z0) = L( (Sbox(z0) << 24 )

T1(z1) = L( (Sbox(z1) << 16 )

T2(z2) = L( (Sbox(z2) << 8 )

T3(z3) = L( Sbox(z3) )

验证：

设A = (a0<<24 | a1<<16 | a2<<8 | a3)

则T(Z)=L(A) = L(a0<<24) xor L(a1<<16) xor L(a2<<8) xor L(a3) （因为L是线性变换，满足加法同态）

因此，我们可以预先计算4个表：

T0\_table[z] = L( (Sbox(z) << 24) )

T1\_table[z] = L( (Sbox(z) << 16) )

T2\_table[z] = L( (Sbox(z) << 8) )

T3\_table[z] = L( Sbox(z) )

每个表大小为256个32位整数（即1KB）。这样，对于每个32位的Z，我们可以拆分成4个字节，然后查4个表并异或，就得到T(Z)。

这样，一轮操作就变成：

Z = X\_{i+1} xor X\_{i+2} xor X\_{i+3} xor rk\_i

T(Z) = T0\_table[z0] xor T1\_table[z1] xor T2\_table[z2] xor T3\_table[z3]

X\_{i+4} = X\_i xor T(Z)

这种查表法每轮需要4次查表和4次异或（包括拆分Z的异或操作，实际上拆分Z不需要额外操作，因为我们可以直接按字节取）。

优化2：使用AESNI指令

AESNI提供了一组用于加速AES的指令。虽然SM4的S盒与AES不同，但我们可以利用AESNI的指令来加速S盒的计算。具体思路是：

利用AESNI的指令（如AESENC, AESDEC等）来模拟SM4的S盒？或者利用AESNI提供的查表指令（如AES\_SBOX）？但AESNI没有直接暴露S盒，而是通过轮函数指令。

另一种方法是：利用AESNI的指令实现一个快速S盒。我们可以将SM4的S盒变换拆分成多个AESNI指令的步骤。然而，有研究表明SM4的S盒可以通过两个AES的S盒和线性变换组合而成（或者通过仿射变换将SM4的S盒映射到AES的S盒）。

具体方法：存在两个仿射变换A和B，使得 SM4\_Sbox(x) = A( AES\_Sbox( B(x) ) )，其中A和B是仿射变换（可逆的线性变换加上常数偏移）。这样，我们可以用AES的S盒加上前后仿射变换来实现SM4的S盒。

一旦我们有了这样的分解，就可以使用AESNI的指令（如\_mm\_aesenclast\_si128）或者直接使用AES的S盒表（但这样又需要查表）？实际上，我们可以使用AESNI的指令来加速AES的S盒计算，然后通过仿射变换得到SM4的S盒。

但是，注意T-table优化已经将S盒和线性变换L合并了。如果我们使用AESNI指令，那么我们可以重新设计T-table的计算：即用AESNI指令快速计算S盒，然后再进行L变换？或者将仿射变换A和B与L变换合并？

然而，由于AESNI指令是作用于128位的（如一次处理16个S盒），我们可以同时计算多个字节的S盒，这样效率更高。因此，我们可以考虑使用AESNI指令并行处理多个字节的S盒，然后再进行L变换（或者将L变换也向量化）。

具体步骤：

- 将输入Z（32位）拆分成4个字节，但如果我们一次处理多个块，那么我们可以将多个块的对应字节放在一起，形成一个128位的向量（16个字节，但我们一次只需要4个字节？实际上，我们可以一次处理4个块（128位\*4=512位），这样每个块提供4个字节，总共16个字节，正好一个128位寄存器。

- 然后，通过仿射变换B（向量化）处理这16个字节，然后应用AES的S盒

优化3：使用GFNI指令

GFNI（Galois Field New Instructions）指令集提供了一些在伽罗华域上的快速操作。 同样，我们可以将GFNI用于整个T函数，从而加速整个轮函数。

优化4：使用AVX512指令集

在支持AVX512的处理器上，我们可以利用512位寄存器同时处理16个SM4块（因为每个块128位，512位可以容纳4个块）。同时，AVX512提供了更多的向量操作，如旋转指令（VPROLD）和异或操作，可以加速线性变换L。

另外，AVX512还提供了更灵活的gather/scatter操作，以及更宽的向量寄存器，有助于并行处理多个块。

另外，对于SM4-GCM，我们需要实现：

- GCM模式，包括计数器模式加密和Galois域乘法（用于认证标签）。

- 优化Galois域乘法（使用查表法、PCLMULQDQ指令等）。

。

运行结果如下：

