*Project4: do your best to optimize SM3 implementation (software)

代码说明：

此项目是实现了 SM3 的优化。

首先，按照老师 PPT 里发的内容完成了 SM3 的基本实现（见 project4），用来与优化后的 SM3 比较，作为对照。

此项目我主要在 sm3_compress(uint32_t digest[8], const unsigned char block[64])函数里实现了优化。优化主要分为两种：

第一种优化方式，是利用了 SIMD 指令集优化。

SIMD，即 Single Instruction, Multiple Data，一条指令操作多个数据．是 CPU 基本指令集的扩展。SIMD 指令集能让一条指令同时对八路数据进行加减乘除、与非或操作，提升运行效率。

在本项目的优化中，以下面这段代码为例说明：

```
__m256i a1 = _mm256_loadu_epi32(&W[0]);   //用 SIMD 指令集优化
__m256i b1 = _mm256_loadu_epi32(&W[4]);
__m256i c1 = _mm256_xor_si256(a1, b1);
_mm256_storeu_epi32(&W1[0], c1);
```

上述代码中使用了 AVX2 的 256 位寄存器__m256i 来处理数据。为了适应内存对齐的要求，我们使用了_mm256_loadu_si256 和_mm256_storeu_si256 函数来加载和存储未对齐的数据。_mm256_xor_si256 是 AVX2 指令集中用于执行两个数据按位异或操作的函数。

利用 AVX2 指令集的 256 位向量处理能力，我同时从数组 W 的两个位置加载数据，然后对数据进行按位异或，最后将结果存储回数组 W1。这样的优化可以提高并行性能，并减少了对内存的多次读写操作，从而提高了代码的执行效率。

第二种，是比较简单的一种，是把 for 循环展开。我把一些循环次数较少的 for 循环直接展开，缩短了运行时间，部分如下图：

```
W[0] = cpu_to_be32(pblock[0]);  //优化：把for循环展开
W[1] = cpu_to_be32(pblock[1]);
W[2] = cpu_to_be32(pblock[2]);
W[3] = cpu_to_be32(pblock[3]);
W[4] = cpu_to_be32(pblock[4]);
W[5] = cpu_to_be32(pblock[5]);
W[6] = cpu_to_be32(pblock[6]);
W[7] = cpu_to_be32(pblock[7]);
W[8] = cpu_to_be32(pblock[8]);
W[9] = cpu_to_be32(pblock[9]);
W[10] = cpu_to_be32(pblock[10]);
W[11] = cpu_to_be32(pblock[11]);
W[12] = cpu_to_be32(pblock[12]);
W[13] = cpu_to_be32(pblock[13]);
W[14] = cpu_to_be32(pblock[14]);
W[15] = cpu_to_be32(pblock[15]);
W[16] = P1(W[0] ^ W[7] ^ ROTATELEFT(W[13], 15)) ^ ROTATELEFT(W[3], 7) ^ W[10];
W[17] = P1(W[1] ^ W[8] ^ ROTATELEFT(W[14], 15)) ^ ROTATELEFT(W[4], 7) ^ W[11];
W[18] = P1(W[2] ^ W[9] ^ ROTATELEFT(W[15], 15)) ^ ROTATELEFT(W[5], 7) ^ W[12];
W[19] = P1(W[3] ^ W[10] ^ ROTATELEFT(W[16], 15)) ^ ROTATELEFT(W[6], 7) ^ W[13];
W[20] = P1(W[4] ^ W[11] ^ ROTATELEFT(W[17], 15)) ^ ROTATELEFT(W[7], 7) ^ W[14];
W[21] = P1(W[5] ^ W[12] ^ ROTATELEFT(W[18], 15)) ^ ROTATELEFT(W[8], 7) ^ W[15];
W[22] = P1(W[6] ^ W[13] ^ ROTATELEFT(W[19], 15)) ^ ROTATELEFT(W[9], 7) ^ W[16];
W[23] = P1(W[7] ^ W[14] ^ ROTATELEFT(W[20], 15)) ^ ROTATELEFT(W[10], 7) ^ W[17];
W[24] = P1(W[8] ^ W[15] ^ ROTATELEFT(W[21], 15)) ^ ROTATELEFT(W[11], 7) ^ W[18];
W[25] = P1(W[9] ^ W[16] ^ ROTATELEFT(W[22], 15)) ^ ROTATELEFT(W[12], 7) ^ W[19];
```

用第一种优化方式（即 SIMD 指令集）的地方也将 for 循环展开了，部分如下图：

```
__m256i a1 = _mm256_loadu_epi32(&W[0]);   //用SIMD指令集优化
__m256i b1 = _mm256_loadu_epi32(&W[4]);
__m256i c1 = _mm256_xor_si256(a1, b1);
_mm256_storeu_epi32(&W1[0], c1);
__m256i a2 = _mm256_loadu_epi32(&W[8]);
__m256i b2 = _mm256_loadu_epi32(&W[12]);
__m256i c2 = _mm256_xor_si256(a2, b2);
_mm256_storeu_epi32(&W1[8], c2);
__m256i a3 = _mm256_loadu_epi32(&W[16]);
__m256i b3 = _mm256_loadu_epi32(&W[20]);
__m256i c3 = _mm256_xor_si256(a3, b3);
_mm256_storeu_epi32(&W1[16], c3);
__m256i a4 = _mm256_loadu_epi32(&W[24]);
__m256i b4 = _mm256_loadu_epi32(&W[28]);
__m256i c4 = _mm256_xor_si256(a4, b4);
_mm256_storeu_epi32(&W1[24], c4);
__m256i a5 = _mm256_loadu_epi32(&W[32]);
__m256i b5 = _mm256_loadu_epi32(&W[36]);
__m256i c5 = _mm256_xor_si256(a5, b5);
_mm256_storeu_epi32(&W1[32], c5);
__m256i a6 = _mm256_loadu_epi32(&W[40]);
__m256i b6 = _mm256_loadu_epi32(&W[44]);
__m256i c6 = _mm256_xor_si256(a6, b6);
_mm256_storeu_epi32(&W1[40], c6);
__m256i a7 = _mm256_loadu_epi32(&W[48]);
__m256i b7 = _mm256_loadu_epi32(&W[52]);
```

实现方式：

用 c++实现

效果：

在自己电脑上 CPU：11 代 i7

未优化代码运行 1000000 次所用时间及 hash 值：

优化后代码运行 1000000 次所用时间及 hash 值：

分工：

自己独立完成