**\*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](https://so.csdn.net/so/search?q=SIMD&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_42817735/article/details/_blank)，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]);

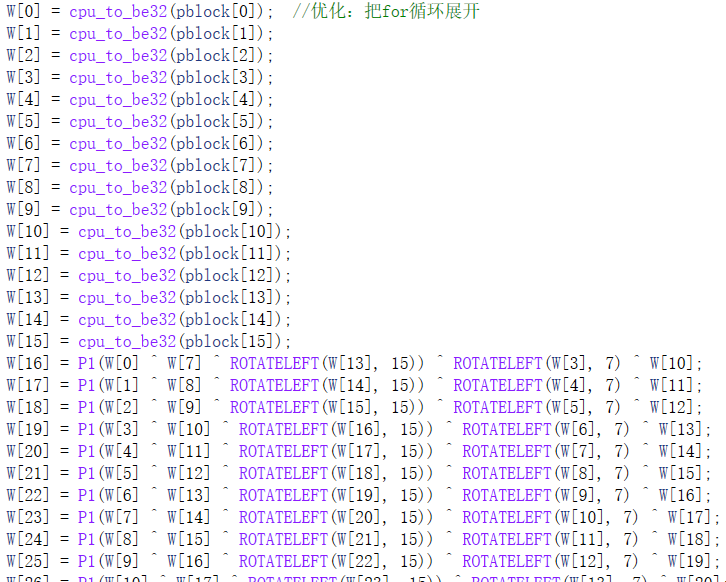
\_\_m256i c1 = \_mm256\_xor\_si256(a1, b1);

\_mm256\_storeu\_epi32(&W1[0], c1);

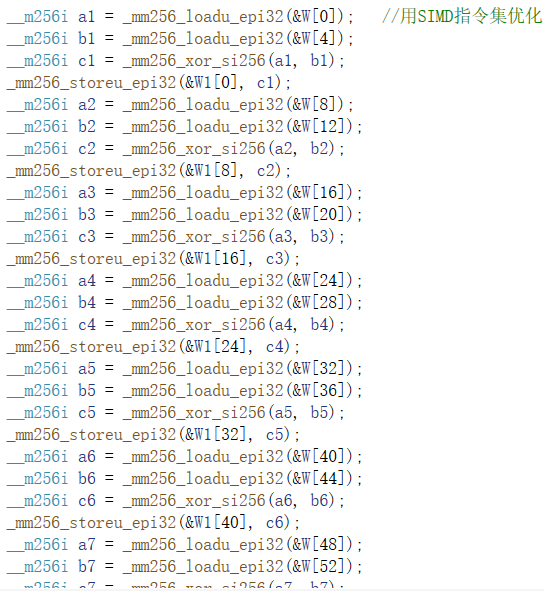
上述代码中使用了AVX2的256位寄存器\_\_m256i来处理数据。为了适应内存对齐的要求，我们使用了\_mm256\_loadu\_si256和\_mm256\_storeu\_si256函数来加载和存储未对齐的数据。\_mm256\_xor\_si256是AVX2指令集中用于执行两个数据按位异或操作的函数。

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：



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：



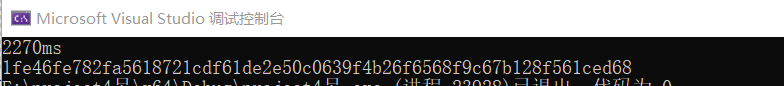
实现方式：

用c++实现

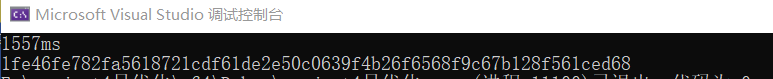
效果：

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