矩阵和向量乘法（采用向量指令）

## 1 原理

在这里，测量的信道会产生一个均衡矩阵，或者说加权矩阵，该矩阵会跟我们的发射数据相乘，这里我就简单的把矩阵和向量写成H和X。这里我以四维的为例。

 （1-1）

这里主要做的是用H的转置与X相乘，公式如下：

 （1-2）

这里主要是做复数相乘（H和X的元素都是复数），那么将会有十六次复数相乘，因此采用我们之前已经做好的规模为16的复数向量相乘的接口做这乘法即可。

我们将数据整合成两个向量，如下所示：

（1-3）

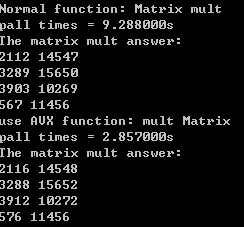
然后调用Mult\_complex32Vector\_2接口（做规模为16的复数相乘的接口），得到结果c[16]，c[16]的每连续四个数的和便是我们的乘积结果，这里调用向量指令的加法接口。

我们把c[16]的位置为1，5，9，13的数据装入寄存器temp1，把c[16]的位置为2，6，10，14的数据装入寄存器temp2，依此类推，一直到temp4，这里的寄存器是128位的，因为四个复数的大小也就是128位的。然后把这四个寄存器依次累加即可。如下：

 （1-4）

最终temp1的四个数据即是我们矩阵和向量相乘的结果，最后用store指令把这几个数据保存到目的地址即可。

## 2 测试

我们随机产生了H和x，然后按照常规方式直接做乘法，和向量指令做乘法。各自运行了1000\_0000万次，分别记录了开始时间和结束时间，然后比较各自的时间即可验证使用向量指令对于速度的提升。我在vs2013环境下面，采用i3处理器，测试结果如下：

当然采用向量指令由于其本来的矢量运算的截位和移位，所以有点误差，这都是能接受的，结果误差范围内是对的。

核心代码片段如下：

产生H和X：

for (i = 0; i<Matrix\_H; i++){

for (j = 0; j < Matrix\_H; j++){

h[i][j].real = (double)(rand() / (double)RAND\_MAX) \* (0x1 << 13);

h[i][j].imag = (double)(rand() / (double)RAND\_MAX) \* (0x1 << 13);

}

x[i].real = (double)(rand() / (double)RAND\_MAX) \* (0x1 << 13);

x[i].imag = (double)(rand() / (double)RAND\_MAX) \* (0x1 << 13);

}

使用常规方法测试时间：(multForMatrix在代码中有定义)

start\_time = clock();

for (times = 0; times < 10000000; times++)

multForMatrix(h, x, x\_dest);

end\_time = clock();

使用向量指令测试时间：

start\_time = clock();

for (times = 0; times < 10000000; times++)

Matrix\_Mult\_AVX2(h, x, x\_dest);

end\_time = clock();