代码优化实验报告

实验目的

1. 通过优化已给出的calculate1函数并实现calculate2函数，使其在输出正确结果的情况下拥有尽可能短的运行时间。
2. 熟悉循环展开等代码优化手段，熟悉SIMD技术。

实验内容

优化后的calculate1函数和实现的calculate2函数

```
void calculate1(int n)
{
    long long sum_qty = 0, sum_base_price = 0;
    double sum_disc_price = 0, sum_charge = 0, sum_discount = 0, avg_qty = 0, avg_price = 0
    int a=0;int limit=Table.num-1;int tnum=Table.num;
    long long sum_qty1=0,sum_qty2=0;
    long long sum_base_price1=0,sum_base_price2=0;
    double sum_discount1=0,sum_discount2=0;
    double sum_disc_price1=0,sum_disc_price2=0;
    double sum_charge1=0,sum_charge2=0;
    for (a = 0; a < limit; a+=2)
    {
        if(Table.date[a]<=n)
        {
            if(Table.date[a+1]<=n)
            {
                sum_qty1 = sum_qty1 + Table.quantity[a];
                sum_qty2 = sum_qty2 + Table.quantity[a+1];
                sum_base_price1 = sum_base_price1 + Table.price[a];
                sum_base_price2 = sum_base_price2 + Table.price[a+1];
                sum_discount1 = sum_discount1 + Table.discount[a];
                sum_discount1 = sum_discount1 + Table.discount[a+1];
                sum_disc_price1 = sum_disc_price1 + Table.price[a] - (Table.price[a] * Tabl
            sum_disc_price2 = sum_disc_price2 + Table.price[a+1] - (Table.price[a+1]*Ta
            sum_charge1 = sum_charge1 + Table.price[a] * (1-Table.discount[a]) * (1+Tab
            sum_charge2 = sum_charge2 + Table.price[a+1] * (1-Table.discount[a+1]) * (1
            count_order+=2;
        }
        else{
            sum_qty1 = sum_qty1 + Table.quantity[a];
            sum_base_price1 = sum_base_price1 + Table.price[a];
            sum_discount1 = sum_discount1 + Table.discount[a];
            sum_disc_price1 = sum_disc_price1 + Table.price[a] - (Table.price[a] * Table
            sum_charge1 = sum_charge1 + Table.price[a] * (1-Table.discount[a]) * (1+Tab
            count_order++;
        }
    }else if(Table.date[a+1]<=n){
            sum_qty2 = sum_qty2 + Table.quantity[a+1];
            sum_base_price2 = sum_base_price2 + Table.price[a+1];
            sum_discount2 = sum_discount2 + Table.discount[a+1];
            sum_disc_price2 = sum_disc_price2 + Table.price[a+1] - (Table.price[a+1]*Ta
            sum_charge2 = sum_charge2 + Table.price[a+1] * (1-Table.discount[a+1]) * (1
            count_order++;
    }
}
sum_qty =sum_qty1+sum_qty2;
sum_base_price=sum_base_price1+sum_base_price2;
sum_discount = sum_discount1 +sum_discount2;
sum_disc_price=sum_disc_price1+sum_disc_price2;
sum_charge=sum_charge1+sum_charge2;
for(; a<tnum ; a++ )
{
    if(Table.date[a]<=n)
    {
        sum_qty = sum_qty +Table.quantity[a];
        sum_base_price = sum_base_price + Table.price[a];
        sum_discount = sum_discount + Table.discount[a];
        sum_disc_price = sum_disc_price + Table.price[a] - (Table.price[a] * Table.disc
        sum_charge = sum_charge + Table.price[a] * (1-Table.discount[a]) * (1+Table.tax
        count_order++;
    }
}
SumQuantity = sum_qty;
SumBasePrice = sum_base_price;
SumDiscPrice = sum_disc_price;
SumCharge = sum_charge;
```

```
        AvgQuantity = SumQuantity / Table.num;
        AvgPrice = SumBasePrice / Table.num;
        AvgDiscount = sum_discount / Table.num;
        total = count_order;
    }


    void calculate2(int n)
    {
        long long sum_qty = 0, sum_base_price = 0;
        double sum_disc_price = 0, sum_charge = 0, sum_discount = 0, avg_qty = 0, avg_price = 0
        int* dat=Table.date;int a=0;int limit=Table.num-7;
        int tnum = Table.num;int* quan=Table.quantity;
        __m256i sum_vec1 = _mm256_setzero_si256();
        int * pric=Table.price;
        __m256i sum_vec2 = _mm256_setzero_si256();
        __m256i count_sum = _mm256_setzero_si256();
        for(a = 0; a < tnum-7; a+=8)
        {
            __m256i quan_vec = _mm256_set_epi32(
            quan[a],quan[a+1],quan[a+2],quan[a+3],quan[a+4],
            quan[a+5],quan[a+6],quan[a+7]
            );
            __m256i dat_vec = _mm256_set_epi32(
            dat[a],dat[a+1],dat[a+2],dat[a+3],dat[a+4],dat[a+5],
            dat[a+6],dat[a+7]
            );
            __m256i mask = _mm256_cmpgt_epi32(dat_vec, _mm256_set1_epi32(n));
            mask = _mm256_andnot_si256(mask, _mm256_set1_epi32(-1));
            quan_vec = _mm256_and_si256(quan_vec, mask);
            sum_vec1 = _mm256_add_epi32(sum_vec1, quan_vec);

            __m256i pric_vec = _mm256_set_epi32(
            pric[a],pric[a+1],pric[a+2],pric[a+3],pric[a+4],
            pric[a+5],pric[a+6],pric[a+7]
            );
            pric_vec = _mm256_and_si256(pric_vec, mask);
            sum_vec2 = _mm256_add_epi32(sum_vec2, pric_vec);

            __m256i count_vector = _mm256_set1_epi32(1);
            count_vector= _mm256_and_si256(count_vector,mask);
            count_sum=_mm256_add_epi32(count_sum,count_vector);
        }
```

```
const int *sum1;
sum1=(const int*)&sum_vec1;
int* sum2;
sum2=(int*)&sum_vec2;
int * sum6;
sum6=(int *)&count_sum;
for (int i = 0; i < 8; ++i) {
    sum_qty += sum1[i];
    sum_base_price += sum2[i];
    count_order+=sum6[i];
}
for(;a<tnum;a++)
{
    if (dat[a] <= n)
    {
        sum_qty += quan[a];
        sum_base_price += pric[a];
        count_order++;
    }
}
double* disc=Table.discount;
__m256d sum_vec3 = _mm256_setzero_pd();
__m256d sum_vec4 = _mm256_setzero_pd();
double* tax=Table.tax;
__m256d sum_vec5 = _mm256_setzero_pd();
for(a = 0; a < tnum-3; a+=4)
{
    __m256i mask_int_vector = _mm256_set_epi32(
    dat[a],dat[a],dat[a+1],dat[a+1],dat[a+2],dat[a+2],
    dat[a+3],dat[a+3]
    );
    __m256i mask = _mm256_cmpgt_epi32(mask_int_vector, _mm256_set1_epi32(n));
    mask = _mm256_andnot_si256(mask, _mm256_set1_epi32(-1));
    __m256d maskd = _mm256_castsi256_pd(mask);
    __m256d data_vec = _mm256_set_pd(disc[a],disc[a+1],disc[a+2],disc[a+3]);
    data_vec = _mm256_and_pd(data_vec, maskd);
    sum_vec3 = _mm256_add_pd(sum_vec3, data_vec);
    __m256d one_vec = _mm256_set1_pd(1.0);
    __m128i int_vector_of4 = _mm_set_epi32(pric[a],pric[a+1],
    pric[a+2],pric[a+3]);
    __m256d double_vector_of4 = _mm256_set_pd(disc[a],disc[a+1],disc[a+2],disc[a+3]);
    __m256d int_to_double_vector_of4 = _mm256_cvtepi32_pd(int_vector_of4);
    int_to_double_vector_of4 = _mm256_and_pd(int_to_double_vector_of4, maskd);
```

```
        double_vector_of4 = _mm256_and_pd(double_vector_of4,maskd);
        __m256d subd_vector = _mm256_sub_pd(one_vec,double_vector_of4);
        __m256d mul_vector = _mm256_mul_pd(int_to_double_vector_of4,subd_vector);
        sum_vec4 = _mm256_add_pd(sum_vec4,mul_vector);
        __m128i int_vector_of5 = _mm_set_epi32(pric[a],pric[a+1],
        pric[a+2],pric[a+3]);
        __m256d double_vector1 = _mm256_set_pd(disc[a],disc[a+1],disc[a+2],disc[a+3]);
        __m256d double_vector2 = _mm256_set_pd(tax[a],tax[a+1],tax[a+2],tax[a+3]);
        __m256i int_mask_vector = _mm256_set_epi32(
        dat[a],dat[a],dat[a+1],dat[a+1],dat[a+2],dat[a+2],
        dat[a+3],dat[a+3]
        );
        __m256d int_to_double_vector_of5 = _mm256_cvtepi32_pd(int_vector_of5);
        int_to_double_vector_of5 = _mm256_and_pd(int_to_double_vector_of5, maskd);
        double_vector1 = _mm256_and_pd(double_vector1,maskd);
        double_vector2 = _mm256_and_pd(double_vector2,maskd);
        double_vector1 = _mm256_sub_pd(one_vec,double_vector1);
        double_vector2 = _mm256_add_pd(one_vec,double_vector2);
        __m256d temp = _mm256_mul_pd(double_vector1,double_vector2);
        temp = _mm256_mul_pd(temp,int_to_double_vector_of5);
        sum_vec5 = _mm256_add_pd(sum_vec5,temp);
    }
    double sum3[4];
    _mm256_storeu_pd(sum3, sum_vec3);
    double *sum4;
    sum4= (double*)&sum_vec4;
    double sum5[4];
    _mm256_storeu_pd(sum5, sum_vec5);
    for (int i = 0; i < 4; ++i) {
        sum_discount += sum3[i];
        sum_disc_price += sum4[i];
        sum_charge += sum5[i];
    }
    for(;a<tnum;a++)
    {
        if (dat[a] <= n)
        {
            sum_discount += disc[a];
            sum_disc_price += pric[a] * (1 - disc[a]);
            sum_charge += pric[a] * (1 - disc[a]) * (1 + tax[a]);
        }
    }
    SumQuantity = sum_qty;
```

```
        SumBasePrice = sum_base_price;
        SumDiscPrice = sum_disc_price;
        SumCharge = sum_charge;
        AvgQuantity = SumQuantity / Table.num;
        AvgPrice = SumBasePrice / Table.num;
        AvgDiscount = sum_discount / Table.num;
        total = count_order;
    }
```

优化和实现过程

对于第一个函数，我按照课本所示思路进行了减少循环中的函数调用、识别结果不会改变的运算移出循环、消除不必要的内存引用和循环展开。具体来说，我将六个循环整合为一个，将getdata这个函数从循环中移除，然后进行了2x2循环展开。

对于第二个函数，我运用了avx指令集中的部分指令，对前两个和最后一个涉及整数的运算进行了八个一组的并行运算，对涉及浮点数的第三到第五个进行了四个一组的并行运算。对于值选择（date大于某个值才能被统计上）的问题，我采用了掩码，将符合条件的位筛选出来再进行运算。

实验结果

经过在本地虚拟机的Linux系统上的运行，相较未优化过的0.602999秒，进行了2x2循环展开的calculate1函数运行时间为0.096940，运用SIMD指令的calculate2函数运行时间为0.3487640秒。在结果上，calculate1在可显示的位数上基本没有差异，calculate2在部分浮点数结果上有1e-10以上的差异。

实验总结

遇到的困难

1. 对循环展开不够熟悉，最开始没能想到几个循环合并为一个的方法。
2. 对SIMD指令不熟悉，在编译选项上遇到困难，最开始使用了-march=skylake-avx512编译选项，之后根据补充实验说明的要求更改了整数向量改成浮点数向量的指令。
3. 最开始使用的是gcc编译，导致必须要加上-lstdc++才能链接成功，后换为g++就解决了。

总结

1. 理解了通过修改函数进行优化的手段。
2. 理解了通过SIMD指令集进行并行的手段。