山东大学 计算机科学与技术 学院

汇编语言 课程实验报告

| | | |
|---|---|---|
| 学号：202200130053 | 姓名：陈红瑞 | 班级：3 班 |
| 实验题目：实验 13：矩阵乘法 | | |
| 实验学时：2 | 实验日期：20241223 | |
| 实验目的： 掌握汇编向量化优化方法。 | | |
| 实验环境：Windows11、DOSBox-0.74、Masm64 | | |
| 源程序清单：<br>lab13.c | | |
| 编译及运行结果：<br>这里首先需要通过文件读取所有的矩阵数据，并保存到数值中，这里定义了 3 个数组，分别用于保存所有的矩阵 A 和矩阵 B 以及所有的矩阵 C。 | | |

```
uint32_t a[16][1024][1024];

uint32_t b[1024][1024];

uint32_t c[16][1024][1024];
```

```
char fname[50];
for (int i = 1; i <= 16; i++) {
    sprintf(fname, "E:/Desktop/vector/mat/A%d.txt", i);
    FILE *fp = fopen(fname, "r");
    for (int j = 0; j < 1024; j++) {
        for (int k = 0; k < 1024; k++) {
            fscanf(fp, "%lu", &a[i-1][j][k]);
            // printf("%llu ", a[i-1][j][k]);
        }
    }
    fclose(fp);
}
```

下面再实现两个函数，分别用于计时和计算校验和（k 值）。这里的计算时间参考了前面的实验中的方法。此外，这里还实现了对 B 进行矩阵转置的方法。

```
uint64_t rdtsc(void)
{
    uint32_t lo, hi;
    __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
    return ((uint64_t)hi << 32) | lo;
}

void checksum() {
    uint64_t sum = 0;
    for (int i = 0; i < 16; i++) {
        for (int j = 0; j < 1024; j++) {
            for (int k = 0; k < 1024; k++) {
                sum += c[i][j][k];
            }
        }
    }
    printf("K = %llu\n", sum);
}
```

```
void transpose() {
    for (int i = 0; i < 1024; i++) {
        for (int j = 0; j < i; j++) {
            uint32_t temp = b[i][j];
            b[i][j] = b[j][i];
            b[j][i] = temp;
        }
    }
}
```

下面再依次实现矩阵乘法，先转置 B 的矩阵乘法，先转置 B 的 AVX2 乘法。

```
void mul1() {
    for (int i = 0; i < 16; i++) {
        for (int j = 0; j < 1024; j++) {
            for (int k = 0; k < 1024; k++) {
                for (int l = 0; l < 1024; l++) {
                    c[i][j][k] += a[i][j][l] * b[l][k];
                }
            }
        }
    }
    // checksum();
}
```

```
void mul2() {
    transpose();
    for (int i = 0; i < 16; i++) {
        for (int j = 0; j < 1024; j++) {
            for (int k = 0; k < 1024; k++) {
                for (int l = 0; l < 1024; l++) {
                    c[i][j][k] += a[i][j][l] * b[k][l];
                }
            }
        }
    }
```

```
void mul3() {
    transpose();
    for (int i = 0; i < 16; i++) {
        for (int j = 0; j < 1024; j++) {
            for (int k = 0; k < 1024; k++) {
                __m256i sum = _mm256_setzero_si256();
                for (int l = 0; l < 1024; l += 8) {
                    __m256i a256 = _mm256_loadu_si256((__m256i*)&a[i][j][l]);
                    __m256i b256 = _mm256_loadu_si256((__m256i*)&b[k][l]);
                    sum = _mm256_add_epi32(sum, _mm256_mullo_epi32(a256, b256));
                }
                uint32_t sum_arr[8];
                _mm256_storeu_si256((__m256i*)sum_arr, sum);
                for (int l = 0; l < 8; l++) {
                    c[i][j][k] += sum_arr[l];
                }
            }
        }
    }
```

在 AVX2 矩阵乘法中,这里使用了 immintrin.h 库,并定义变量__m256i 表示一个能存储 256 位整型的向量，然后使用指令_mm256_loadu_si256

实现加载数据到向量中，由于这里矩阵保存的数字是 32 位，因此每个向量能处理 256/32 = 8 个数字，因此在实现矩阵乘法中，能通过 AVX2 一次处理 8 个数的乘法。这里还使用了_mm256_mullo_epi32 实现向量的乘法，然后使用_mm256_add_epi32 实现将每 32 位数字相加，得到当前 8 个元素各自相乘后的和。最后将这个数字加到当前正在计算的矩阵 C 中。

最后分别将这三种方法进行测试，这里定义了函数，使用 rdtsc 实现计时，其中第 2，3 种方法种还需要将矩阵转置操作记录在时间里面，当第二种方法计算完成后，还需要再转置，还原矩阵 B，然后在第三种方法中再计算转置与矩阵乘法。每一种方法计算完成后，先计算 K 值，然后输出时间。

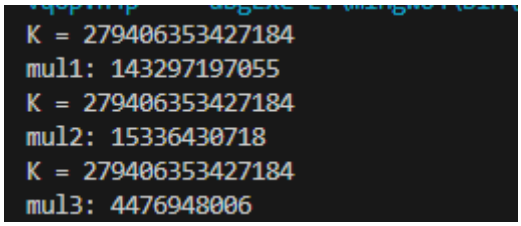
```
void test() {
    uint64_t start = rdtsc();
    mul1();
    uint64_t end = rdtsc();
    checksum();
    printf("mul1: %llu\n", end - start);

    clear();
    start = rdtsc();
    mul2();
    end = rdtsc();
    checksum();
    printf("mul2: %llu\n", end - start);
    transpose();

    clear();
    start = rdtsc();
    mul3();
    end = rdtsc();
    checksum();
    printf("mul3: %llu\n", end - start);
}
```

下面是程序运行的结果。

其中第一种方法速度最慢，第二种方法由于先转置了 B，用时几乎为第一种方法的 1/10，而第三种方法用时最少。




问题及收获：

1. 通过这个实验了解 AVX2 乘法的实现方法，这里使用了 immintrin.h 库实现了 AVX2 指令种的一些相关函数调用，包括加法，乘法，加载向量等，实现了速度更快的矩阵乘法，并通过测试可以看出与前面的方法相比速度有明显的优势。