Exp 0 报告

方科晨　　2021013400

1 代码

1.1 openmp_pow

修改后的 pow_a 函数的源代码如下：

```
void pow_a(int *a, int *b, int n, int m) {
    // TODO: 使用 omp parallel for 并行这个循环
        #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        int x = 1;
        for (int j = 0; j < m; j++)
            x *= a[i];
        b[i] = x;
    }
}
```

1.2 mpi_pow

修改后的 pow_a 函数的源代码如下：

```
void pow_a(int *a, int *b, int n, int m, int comm_sz /* 总进程数 */) {
    // TODO: 对这个进程拥有的数据计算 b[i] = a[i]^m
        for (int i = 0; i < n / comm_sz; i++)
        {
                int x = 1;
                for (int j = 0; j < m; j++)
                {
                        x *= a[i];
                }
                b[i] = x;
        }
}
```

2 openmp 版本性能

在 $n = 112000, m = 100000$ 下，使用 $1, 7, 14, 28$ 线程的运行时间分别是：$T_1 = 14015212\mu s, T_7 = 2020439\mu s, T_{14} = 1021353\mu s, T_{28} = 522389\mu s$
可以求得相对于单线程的加速比分别为：$S(1) = 1, S(7) = 6.9367, S(14) = 13.7222, S(28) = 26.8291$ ，可以看出非常接近线性加速。

3 mpi 版本性能

在 $n = 112000, m = 100000$ 下，使用 $1 \times 1, 1 \times 7, 1 \times 14, 1 \times 28, 2 \times 28$ 进程下运行时间分别为 $T_1 = 14017820\mu s, T_7 = 2012535\mu s, T_{14} = 1006988\mu s, T_{28} = 502862\mu s, T_{56} = 403601\mu s$
可以求得相对于单线程的加速比分别为：$S(1) = 1, S(7) = 6.9653, S(14) = 13.9205, S(28) = 27.8761, S(56) = 34.7319$ ，可以看出，在单机上，加速比可以近似线性加速，但在多机上，由于数据交换等原因，速度的增加幅度明显小于进程数的增加幅度。