实验报告: MPI 和 OpenMP 并行化 pow_a

赵袭明

2021012319

2025 年 3 月 10 日

1 实验任务

使用 MPI 和 OpenMP 并行化下述代码，代码的作用是计算 $b[i] = a[i]^m$ 其中 $a[i]$ 和 $b[i]$ 是两个长度为 n 的数组。

```
void pow_a(int *a, int *b, int n, int m) {
    for (int i = 0; i < n; i++) {
        int x = 1;
        for (int j = 0; j < m; j++)
            x *= a[i];
        b[i] = x;
    }
}
```

2 代码实现

2.1 OpenMP 版本

修改后的 pow_a 函数如下:

```
#include <omp.h>
void pow_a(int *a, int *b, int n, int m) {
    #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;
    }
}
```

只需加入一行指导语句即可。

2.2 MPI 版本

修改后的 pow_a 函数如下:

```
#include <mpi.h>
void pow_a(int *a, int *b, int n, int m, int comm_sz) {
    int local_n = n / comm_sz;
    for (int i = 0; i < local_n; i++) {
        int x = 1;
        for (int j = 0; j < m; j++)
            x *= a[i];
        b[i] = x;
    }
}
```

在 pow_a 函数内, 每个进程只计算属于自己的 a 数组部分，对 $a[i]$ 进行 m 次方运算并存储到 $b[i]$ 即可。

3 实验结果

原始的代码输出如下:

```
g++ openmp_pow.cpp -O3 -std=c++11 -fopenmp -o openmp_pow
mpicxx mpi_pow.cpp -O3 -std=c++11 -o mpi_pow
openmp_pow: n = 112000, m = 100000, thread_count = 1
Congratulations!
Time Cost: 14015005 us
```

```
openmp_pow: n = 112000, m = 100000, thread_count = 7
Congratulations!
Time Cost: 14018343 us

openmp_pow: n = 112000, m = 100000, thread_count = 14
Congratulations!
Time Cost: 14008828 us

openmp_pow: n = 112000, m = 100000, thread_count = 28
Congratulations!
Time Cost: 14009171 us

mpi_pow: n = 112000, m = 100000, process_count = 1
Wrong answer at position 34133: 0 != -259604863
srun: error: conv1: task 0: Exited with exit code 1
```

3.1 OpenMP 版本实验结果

实验环境：$n = 112000, m = 100000$

使用 OpenMP 和 MPI 修改后的到了如下输出:

```
g++ openmp_pow.cpp -O3 -std=c++11 -fopenmp -o openmp_pow
mpicxx mpi_pow.cpp -O3 -std=c++11 -o mpi_pow
openmp_pow: n = 112000, m = 100000, thread_count = 1
Congratulations!
Time Cost: 14011480 us

openmp_pow: n = 112000, m = 100000, thread_count = 7
Congratulations!
Time Cost: 2013215 us

openmp_pow: n = 112000, m = 100000, thread_count = 14
```

```
Congratulations!
Time Cost: 1010784 us

openmp_pow: n = 112000, m = 100000, thread_count = 28
Congratulations!
Time Cost: 516337 us

mpi_pow: n = 112000, m = 100000, process_count = 1
Congratulations!
Time Cost: 14009948 us

mpi_pow: n = 112000, m = 100000, process_count = 7
Congratulations!
Time Cost: 2010744 us

mpi_pow: n = 112000, m = 100000, process_count = 14
Congratulations!
Time Cost: 1005803 us

mpi_pow: n = 112000, m = 100000, process_count = 28
Congratulations!
Time Cost: 501768 us

mpi_pow: n = 112000, m = 100000, process_count = 56
Congratulations!
Time Cost: 370754 us

All done!
```

线程数	运行时间 (us)	加速比
1	14009948	1
7	2013215	6.96
14	1010784	13.86
28	516337	27.14

表 1: OpenMP 实验结果

4 运行时间比较

可以看到基本实现了随线程数的线性加速

$N \times P$	运行时间 (us)	加速比
1×1	14011480	1
1×7	2010744	6.97
1×14	1005803	13.93
1×28	501768	27.92
2×28	370754	37.79

表 2: MPI 实验结果

MPI 的实验结果在机器数 $N = 1$ 时基本实现了线性的加速比, 在 $N = 2$ 时加速比偏离了线性, 可能是因为机器间的通信消耗了较多的时间。