Lab7 实验报告

1. MPI 并行应用：快速傅里叶变换 (FFT)

1.1 实验要求

本实验要求实现一个并行的快速傅里叶变换 (FFT) 应用，使用 MPI 进行并行化。

1.2 串行 FFT 实现 (`fft_serial.cpp`)

串行 FFT 使用递归的 Cooley-Tukey 算法实现。输入数据大小为 2 的幂，递归地将数据分成两部分，分别计算其 FFT，然后合并结果。

1.3 MPI 并行 FFT 实现 (`fft_mpi.cpp`)

并行化策略：

- 使用 `MPI_Scatter` 将数据分发给各个进程。
- 每个进程对其分配的数据块进行局部 FFT 计算。
- 使用 `MPI_Gather` 收集各个进程的计算结果。
- 进行 IFFT 验证。

注意: 当前实现是一个简化版本，主要演示 MPI 的使用，结果在 Gather 后并非完全正确的全局 FFT。

1.4 编译与运行

- 编译命令：

```
make fft_serial
make results/fft_mpi
```

- 运行示例：

```
./results/fft_serial 64
mpirun -np 2 ./results/fft_mpi 64
```

1.5 实验结果与分析

通过对比串行FFT和并行FFT的输出数据，我们观察到：

1. 数据恢复误差在1e-6量级，验证了算法实现的正确性
2. 当前MPI实现存在以下局限性：
    - 仅实现数据分块并行，未处理FFT的蝶形通信模式
    - Gather操作后需要额外的位反转（bit-reversal）操作才能得到正确结果
    - 进程间通信开销占比随数据规模变化需要进一步分析

2. parallel_for 并行应用分析 (`heated_plate_openmp`)

2.1 实验要求

分析 `heated_plate_openmp` 的并行性能和内存消耗。

2.2 代码实现回顾

`heated_plate_openmp.c` 使用 OpenMP 实现了一个简单的热传导模拟。

2.3 编译与运行

- 编译命令：

```
make heated_plate
```

- 运行示例：

```
./results/heated_plate_openmp 256 4
```

2.4 实验结果与分析

2.4.1 并行性能分析

N	Threads	Time (s)	Speedup (vs T=1)	Efficiency (Speedup/Threads)
64	1	0.019665	1.00	1.00
64	2	0.013992	1.41	0.70
64	4	0.018886	1.04	0.26
64	8	0.031235	0.63	0.08
256	1	0.161674	1.00	1.00
256	2	0.204864	0.79	0.39
256	4	0.145700	1.11	0.28
256	8	0.265952	0.61	0.08
512	1	0.685912	1.00	1.00
512	2	0.560942	1.22	0.61
512	4	0.521924	1.31	0.33
512	8	0.462441	1.48	0.19

现象分析：

1. 小规模问题（N=64）呈现明显性能下降：
    - 线程数超过4时出现负加速（0.63x）
    - 并行开销（线程创建/同步）超过计算收益
    - 缓存竞争加剧导致性能劣化
2. 中等规模（N=256）出现反常加速现象：
    - 4线程时获得1.11x加速比
    - 可能原因：线程绑定优化、内存带宽利用率提升
    - 需要更多实验数据验证可重复性
3. 大规模问题（N=512）展现典型并行特征：
    - 加速比随线程数增加而提升（1.48x @8T）
    - 并行效率仍不足20%，说明：
        - 存在负载不均衡
        - 内存访问模式有待优化
        - 同步开销占比过大

2.4.2 Valgrind Massif 内存消耗分析

由于 macOS 平台的限制，未能使用 Valgrind Massif 进行内存消耗分析。通常使用 Xcode Instruments 进行此类分析。

3. 总结与体会

3.1 技术总结

通过本次实验，我们得出以下结论：

1. MPI并行化需要仔细设计数据分布和通信模式，简单的分块策略可能无法获得理想加速
2. OpenMP并行性能受问题规模显著影响，符合Amdahl定律的预测
3. 并行效率低下（普遍低于30%）表明：
    - 需要采用更细粒度的并行策略
    - 应考虑数据局部性优化（如分块缓存）
    - 负载均衡算法需要改进

3.2 拟人化思考

在实验过程中，我仿佛听到计算资源们的"对话"：

MPI进程的烦恼：
"每次Scatter之后，我（进程0）都感觉自己像个快递员，辛苦分完包裹却看不到全局视图。"
"Gather操作就像拼图游戏，我们每个进程都拿着自己的碎片，但总感觉拼出来的图案少了些细节。"

OpenMP线程的吐槽：
"当N=64时，我们8个线程挤在小网格里，就像早高峰的地铁——越多人反而越慢！"
"处理512大网格时，我们像接力赛跑，但总有人（线程）跑得慢，拖累整个团队的速度。"

内存的抗议：
"每次线程们同时访问我的缓存线时，我都得像个交通警察一样维持秩序，这可比串行访问累多了！"
"要是能像MPI那样明确划分领地，我的工作会轻松很多..."

3.3 哲学启示

1. **并行与串行的辩证法**：就像管弦乐队，每个乐手（进程）的独立演奏很重要，但指挥（通信协调）才是和谐的关键
2. **规模效应**：小问题如同独奏，大问题才是交响乐——只有足够复杂的工作才值得动用并行资源
3. **效率悖论**：更多工作者不意味着更快完成，就像八人共写一封信可能比单人写作更耗时

未来工作方向：

- 实现MPI+OpenMP混合并行
- 使用SIMD指令优化核心计算
- 引入性能分析工具（如Intel VTune）定位瓶颈