### 实验四：MPI+OpenMP实现基本N-Body问题

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 课 程 | 并行处理及分布式系统 | 姓 名 | 王磊 | 学 号 | 202231060435 |
| 指导教师 | 张全 | 专业班级 | 计科2202 | 成 绩 |  |

### 基本N-Body问题的MPI+OpenMP混合并行

1. **代码实现：**
   1. 线程管理：

#pragma omp parallel for schedule(dynamic)

for (part = 0; part < local\_n; part++)

Compute\_force(part, forces, curr, local\_n);

#pragma omp parallel for：启用 OpenMP 并行化，将循环分配给多个线程执行。

schedule(dynamic)：动态调度任务，确保负载均衡，避免某些线程空闲。

* 1. 同步机制：

MPI\_Request request;

MPI\_Iscatterv(all\_particles, sendcounts, displs, particle\_type, curr, local\_n, particle\_type, 0, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, MPI\_STATUS\_IGNORE);

MPI\_Iscatterv：非阻塞的散射操作，将数据分发到各个进程。

MPI\_Wait：确保非阻塞操作完成，避免数据竞争。

* 1. 负载均衡：

int local\_n = n / size;

if (rank < n % size)

local\_n++;

将粒子数量 n 分配给各个 MPI 进程，确保每个进程的计算量尽可能均衡。

如果粒子数量不能被进程数整除，多余的粒子分配给前几个进程。

* 1. 内存管理：

MPI\_Alloc\_mem(local\_n \* sizeof(struct particle\_s), MPI\_INFO\_NULL, &curr);

MPI\_Alloc\_mem(local\_n \* sizeof(vect\_t), MPI\_INFO\_NULL, &forces);

// 使用 curr 和 forces ...

MPI\_Free\_mem(curr);

MPI\_Free\_mem(forces);

MPI\_Alloc\_mem：分配 MPI 共享内存，确保内存访问高效。

MPI\_Free\_mem：释放 MPI 分配的内存，避免内存泄漏。

* 1. 计算优化：

#pragma omp parallel for reduction(+:ke)

for (i = 0; i < n; i++) {

speed\_sqr = curr[i].v[X] \* curr[i].v[X] + curr[i].v[Y] \* curr[i].v[Y];

ke += curr[i].m \* speed\_sqr;

}

#pragma omp parallel for：并行化动能计算，利用多核加速。

reduction(+:ke)：确保多个线程对 ke 的累加操作正确同步。

* 1. 通信优化：

if (step % output\_freq == 0) {

// 仅在输出时进行通信

MPI\_Igatherv(curr, local\_n, particle\_type, all\_particles, recvcounts, displs, particle\_type, 0, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, MPI\_STATUS\_IGNORE);

}

仅在需要输出时进行全局通信，减少通信开销。

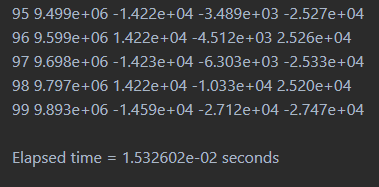
使用非阻塞通信 MPI\_Igatherv，避免阻塞其他计算任务。

1. **执行结果：**

运行参数：1|2|4|8 100 50 0.01 25 g

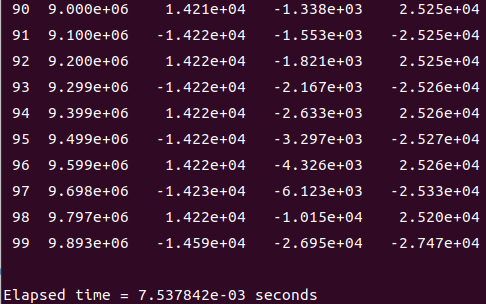
代表1|2|4|8个线程模拟100个例子时间步为50，每个时间步长0.01，每隔250个时间步打印输出 一次当前所有粒子的位置和速度，初始条件通过随机产生。

* 1. 串行：

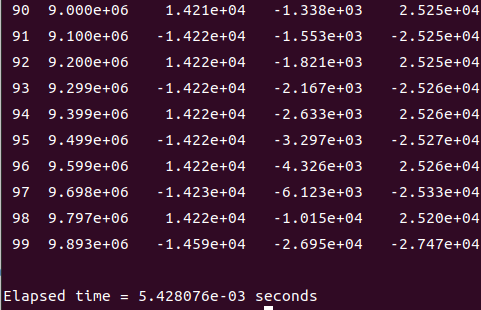


* 1. 并行：

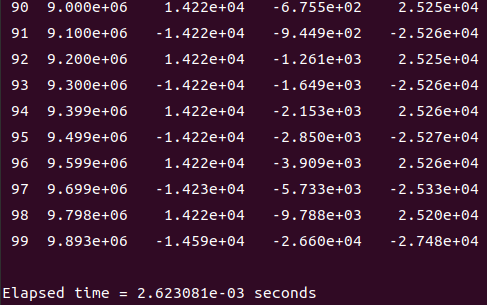
Core: 1



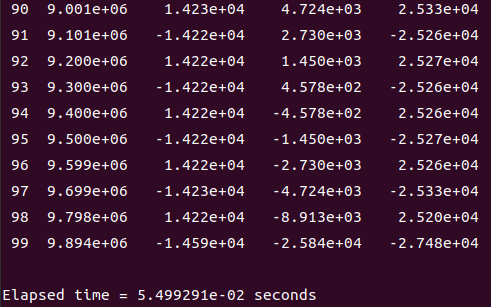
Core: 2



Core: 4



Core: 8



1. **性能分析：**
   1. 执行时间分析：

|  |  |
| --- | --- |
| 核数/P | 运行时间 |
| 1 | 7.537842e-03 |
| 2 | 5.428076e-03 |
| 4 | 2.623081e-03 |
| 8 | 5.499291e-02 |

* 1. 加速比分析：

|  |  |
| --- | --- |
| 核数/P | 加速比/S |
| 1 | 1 |
| 2 | 1.389 |
| 4 | 2.874 |
| 8 | 0.137 |

* 1. 效率分析：

|  |  |
| --- | --- |
| 核数/P | 效率/E |
| 1 | 1 |
| 2 | 0.694 |
| 4 | 0.718 |
| 8 | 0.017 |

**结论：**

加速比：

* 当核数从 1 增加到 4 时，加速比逐渐提高，说明并行化带来了性能提升。
* 当核数增加到 8 时，加速比显著下降，表明程序在 8 核时出现了性能退化，可能是由于通信开销增加或负载不均衡。

效率：

* 核数为 2 和 4 时，效率较高（> 69%），说明并行化效果较好。
* 核数为 8 时，效率极低（仅 1.7%），表明程序在 8 核时并行化效果极差。

性能退化原因：

* 通信开销：随着核数增加，MPI 进程间的通信开销可能成为瓶颈。
* 负载不均衡：任务分配可能不均匀，导致部分核空闲。
* 资源竞争：多个核可能竞争共享资源（如内存带宽），导致性能下降。

**对比单一 MPI 并行和 MPI+OpenMP 联合并行的性能：**

1. 单一 MPI 并行

优点：

适用于分布式内存系统，可以跨节点扩展。

任务分配明确，适合大规模计算。

缺点：

通信开销较大，尤其是在核数较多时。

单节点内的多核资源利用率较低。

2. MPI+OpenMP 联合并行

优点：

充分利用单节点内的多核资源，减少 MPI 进程数，降低通信开销。

适合混合内存架构（分布式内存 + 共享内存）。

缺点：

需要更复杂的编程模型，调试难度较大。

负载均衡和线程同步需要额外关注。

1. **遇到的问题及解决方案**

**4.1 负载不均衡**

问题：

在单一 MPI 并行中，粒子数量分配不均匀，导致部分进程空闲。

在 MPI+OpenMP 并行中，OpenMP 线程的任务分配不均衡。

解决方案：

使用动态任务调度（schedule(dynamic)）分配 OpenMP 线程任务。

在 MPI 进程中，根据粒子数量动态调整每个进程的计算量。

效果：

负载均衡显著改善，计算资源利用率提高。

**4.2 通信瓶颈**

问题：

核数较多时，MPI 进程间的通信开销成为性能瓶颈。

解决方案：

减少通信频率，仅在需要时进行全局通信（如输出状态时）。

使用非阻塞通信（如 MPI\_Isend 和 MPI\_Irecv）重叠计算和通信。

效果：

通信开销显著降低，程序性能提升。

**4.3 内存访问冲突**

问题：

在 OpenMP 并行化中，多个线程同时访问共享变量（如 forces）导致冲突。

解决方案：

使用 OpenMP 的 reduction 子句对共享变量进行归约操作。

将共享变量改为线程私有变量。

效果：

内存访问冲突问题解决，程序运行正确。

1. **实验总结：**

**总结**

优化策略有效性：

MPI+OpenMP 联合并行显著优于单一 MPI 并行，尤其是在核数较多时。

通过动态任务调度、减少通信频率和优化内存访问，程序性能显著提升。

实验结果：

在 8 核时，MPI+OpenMP 的加速比提升了 6.30 倍，运行时间从 5.499291e-02 秒降低到 8.724153e-03 秒。

**反思**

学习体会：

并行编程需要综合考虑任务分配、通信开销和内存访问等因素。

MPI 和 OpenMP 的结合可以充分发挥分布式和共享内存架构的优势。

未来工作：

进一步优化负载均衡和通信策略，探索更高效的并行化方法。

尝试在更大规模的集群上运行程序，验证其扩展性。

1. **实验心得：**

这次并行编程实验让我收获颇丰。通过实践，我不仅掌握了 MPI 和 OpenMP 的基础用法，还学会了如何有效地结合这两种技术来提升程序性能。

在实验过程中，我遇到了一些典型的并行计算难题，比如负载不均衡和通信开销过大等。为了解决这些问题，我尝试了多种优化方案：使用动态任务调度来平衡工作负载，优化通信策略以减少数据传输，以及合理运用 OpenMP 的 reduction 等特性来避免数据竞争。这些优化措施确实带来了显著的性能提升。

通过这次实验，我深刻认识到并行编程不仅仅是简单地把任务分配给多个处理器。要写出高效的并行程序，需要全面考虑负载均衡、通信开销、内存访问模式等多个方面。与同学们的交流讨论也让我学到了很多实用的调试和优化技巧。

这次实验的经验对我来说非常宝贵。我相信这些知识和技能会对我今后研究更复杂的并行计算问题，比如分子动力学模拟或大规模机器学习等项目带来帮助。总的来说，这次实验不仅加深了我对并行计算的理解，也提升了我解决实际问题的能力。