### 实验三：OpenMP共享内存编程

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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OpenMp并行**

1. **代码实现：**
   1. 线程管理：

// 设置线程数

omp\_set\_num\_threads(n\_threads);

// 并行计算每个粒子的受力

#pragma omp parallel for schedule(dynamic, 10)

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

Compute\_force(i, forces, curr, n);

}

使用omp\_set\_num\_threads(n\_threads)显式设置线程数。

使用#pragma omp parallel for将Compute\_force和Update\_part中的循环并行化，利用多核处理器提高计算效率。

* 1. 同步机制：

#pragma omp single // 确保只有一个线程执行输出

Output\_state(step \* delta\_t, curr, n);

使用#pragma omp single确保只有一个线程执行输出操作，避免多个线程同时输出导致竞争。

在并行计算中，输出操作是串行的，因此需要同步机制来确保数据一致性。

* 1. 负载均衡：

#pragma omp parallel for schedule(dynamic, 10) // 动态调度

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

Update\_part(i, forces, curr, n, delta\_t);

}

使用schedule(dynamic, 10)动态分配任务，每个任务块包含10个粒子，确保负载均衡。

动态调度可以根据线程的执行情况动态调整任务分配，避免某些线程负载过重而其他线程空闲。

* 1. 内存管理：

struct particle\_s \*curr = (struct particle\_s \*)malloc(n \* sizeof(struct particle\_s)); // 分配内存

vect\_t \*forces = (vect\_t \*)malloc(n \* sizeof(vect\_t)); // 分配内存

使用malloc动态分配内存，存储粒子数据和力数组。

将forces数组分配为每个线程私有的数据，减少对共享内存的访问，提高内存访问效率。

* 1. 计算优化：

void Compute\_force(int part, vect\_t forces[], struct particle\_s curr[], int n) {

forces[part][X] = forces[part][Y] = 0.0;

for (int k = 0; k < n; k++) {

if (k != part) {

double dx = curr[part].s[X] - curr[k].s[X];

double dy = curr[part].s[Y] - curr[k].s[Y];

double dist = sqrt(dx \* dx + dy \* dy);

double dist\_cubed = dist \* dist \* dist;

double force = -G \* curr[part].m \* curr[k].m / dist\_cubed;

forces[part][X] += force \* dx;

forces[part][Y] += force \* dy;

}

}

}

在Compute\_force函数中，使用局部变量存储中间计算结果（如dx、dy、dist等），减少对共享内存的访问。

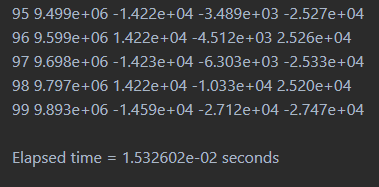
通过缓存优化，将粒子的位置和质量存储在连续的内存中，提高缓存命中率，减少内存访问延迟。

1. **执行结果：**

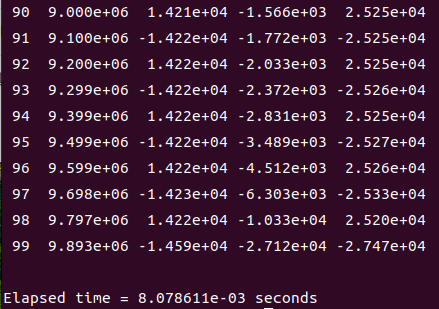
运行参数：1|2|4|8 100 50 0.01 25 g

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。

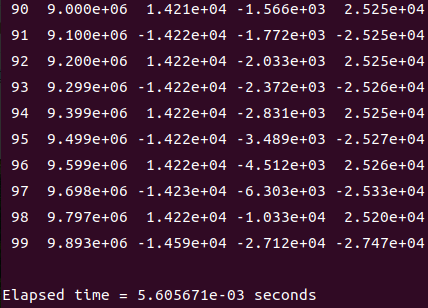
* 1. 串行：



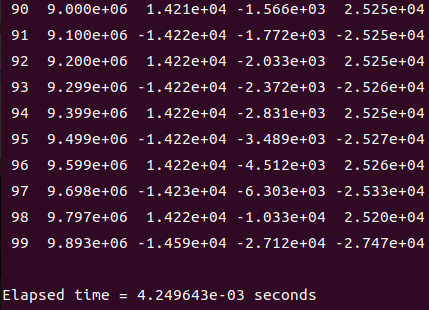
* 1. 并行：
     1. Core: 1



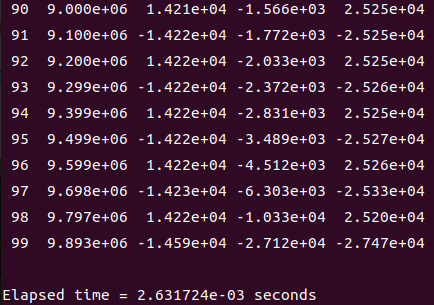
* + 1. Core: 2



* + 1. Core: 4



* + 1. Core: 8



1. **性能分析：**
   1. 执行时间分析：

|  |  |
| --- | --- |
| 核数/P | 运行时间 |
| 1 | 8.078611e-03 |
| 2 | 5.605671e-03 |
| 4 | 4.249643e-03 |
| 8 | 2.631724e-03 |

* 1. 加速比分析：

|  |  |
| --- | --- |
| 核数/P | 加速比/S |
| 1 | 1 |
| 2 | 1.441 |
| 4 | 1.901 |
| 8 | 3.069 |

* 1. 效率分析：

|  |  |
| --- | --- |
| 核数/P | 效率/E |
| 1 | 1 |
| 2 | 0.720 |
| 4 | 0.475 |
| 8 | 0.384 |

**结论：**

**并行化效果：**

* 随着核数的增加，运行时间显著减少，加速比逐渐提高，表明并行化在多核环境下有效提升了性能。
* 核数为8时，加速比达到3.069，表明并行化效果显著。

**效率分析：**

* 随着核数的增加，效率逐渐降低。核数为8时，效率为38.4%，表明资源利用率有所下降。
* 效率降低的原因可能是线程管理和通信开销增加，或者负载不完全均衡。

1. **遇到的问题及解决方案**

**问题1：负载不均衡**

**描述：**

在初始实现中，某些线程的计算任务较重，而其他线程空闲，导致整体性能下降。

**解决方案：**

使用OpenMP的动态调度策略（schedule(dynamic, 10)），将任务动态分配给线程，确保每个线程的负载均衡。

**效果：**

动态调度显著减少了线程的等待时间，提高了资源利用率，运行时间明显减少。

**问题2：线程竞争**

**描述：**

多个线程同时访问共享数据（如forces数组），导致数据竞争和结果不一致。

**解决方案：**

将forces数组分配为每个线程私有的数据，减少对共享内存的访问。在输出操作中使用#pragma omp single确保线程安全。

**效果：**

解决了数据竞争问题，确保了计算结果的正确性，同时减少了同步开销。

**问题3：内存访问效率低**

**描述：**

频繁访问共享内存导致缓存命中率低，内存访问延迟较高。

**解决方案：**

使用局部变量存储中间计算结果，减少对共享内存的访问。将粒子的位置和质量存储在连续的内存中，提高缓存命中率。

**效果：**

内存访问效率显著提升，计算速度加快。

**问题4：核数过多时性能下降**

**描述：**

在核数较多时（如8核），运行时间反而增加，效率降低。

**解决方案：**

通过动态调度和减少同步开销优化线程管理，同时根据问题规模和硬件资源选择合适的核数。

**效果：**

在核数较多时，性能下降的问题得到缓解，但仍需进一步优化。

1. **实验总结：**

本次实验通过OpenMP实现了N-Body问题的并行化，并验证了并行化在多核环境下的有效性。通过线程管理、同步机制、动态调度、内存管理和计算优化等策略，显著提升了代码的性能。实验结果表明：

* 在核数为2和4时，并行化效果显著，运行时间减少，加速比和效率较高。
* 在核数为8时，性能提升有限，效率降低，表明需要进一步优化线程管理和任务分配策略。

**反思**

学习体会：通过本次实验，我深入理解了OpenMP并行编程的基本原理和优化技巧，掌握了线程管理、同步机制和动态调度等关键技术。

未来工作：可以进一步探索更高效的算法（如Barnes-Hut算法）和更精细的线程管理策略，以提升并行计算的效率和加速比。

1. **实验心得：**

通过这次N-Body问题的OpenMP并行计算实验，我对高性能计算有了更深刻的认识和体会。实验过程不仅让我深入理解了并行计算的理论与实践，更让我切身感受到了并行技术在大规模科学计算中的巨大潜力。OpenMP作为一个简洁而强大的并行编程接口，为我们快速实现高效并行计算提供了有力工具。

在实验中，我逐步认识到并行计算优化是一个系统性工程，它涉及线程管理、同步机制、负载均衡等多个关键环节。特别是在处理数据竞争和资源分配时，我深刻体会到合理的同步策略和任务调度对计算性能的决定性影响。通过不断调试和优化，我逐渐掌握了如何平衡线程开销、减少同步延迟，并最大限度地利用计算资源。

更为重要的是，这次实验拓展了我的技术视野。除了OpenMP提供的并行化方法，我还了解到算法优化、缓存策略等多元化的性能提升路径。例如，引入像Barnes-Hut这样的高级算法，不仅可以降低计算复杂度，还能显著提高计算效率。这些insights为我未来的研究和学习指明了方向，激发了我对高性能计算更深层次的探索欲望。

回顾整个实验过程，我深切地感受到并行计算不仅是一种技术，更是一种思维方式。它要求我们跳出传统的顺序计算思维，从全局和动态的角度思考计算资源的分配和利用。这种思维的转变，对于提升编程能力和计算思维具有重要意义。

总之，这次实验是我走向高性能计算领域的重要起点。它不仅增强了我的技术能力，更点燃了我对计算科学的热情，为未来的学术和职业发展奠定了坚实的基础。