**《并行计算》课程实验报告**

**实验3：基于华为云环境的MPI并行编程**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 刘子康 | | 院系 | | 计算学部 | | | 学号 | | 2022113416 | |
| 任课教师 | | 张伟哲 | | | | 指导教师 | 郝萌 | | | | |
| 实验地点 | | 格物213 | | | | 实验时间 | 2024.12.11 | | | | |
| 实验课表现 | | 出勤、表现得分 | |  | | 实验报告  得分 |  | | 实验总分 | |  |
| 操作结果得分 | |  | |
| **一、实验目的** | | | | | | | | | | | |
| 1.了解华为云环境的使用过程；  2.掌握利用华为云环境搭建小型集群环境的过程；  3.掌握MPI程序设计的基本编写、编译与运行方法；  4.了解集群环境下N体问题的并行程序设计方法；  5.掌握利用加速比、运行时间、效率等测度分析并行程序性能；  6.掌握素数计算程序的基本原理；  7.掌握串行素数计算程序的MPI并行优化方法。 | | | | | | | | | | | |
| **二、实验内容** | | | | | | | | | | | |
| **2.1配置基于华为云的MPI并行集群环境**  购买3台华为云的弹性云服务器ECS，实例CPU架构选择鲲鹏计算，操作系统镜像选择openEuler，系统盘至少40GB。    使用wget gist.gitmirror.com/GoldenPotato137/4ae8ccb7cc1152d5e902a3b07aafda46/raw/b5efb7d759d21247bc068eff652aba9d26bb55c5/deploy.sh命令，一键部署环境脚本配置MPI并行环境，给deploy.sh脚本添加权限并运行，输入3台主机的私有IP和主机名称，创建用户名。  **2.2 N体问题MPI并行程序剖析与性能分析**  通过计算机仿真求解N体问题的近似解。  串行算法外层的for循环计算每一个时间步，在每个时间步内n个物体更新了速度和位置；内层的两个for循环计算每一个物体所受到的力及其新的位置和速度。  并行算法使用SPMD计算模型，外层for循环计算每一个时间步，在每个时间步内所有参与的进程并行执行，所有进程都结束时所有n个物体速度和位置都被更新了。在外层for的循环体内，首先计算所有物体对本地物体的作用力，据此更新关于本地物体的信息；内层for循环使得每个进程都获得了所有物体的更新后的信息，实现了进程之间的合作，为下一次迭代做好准备；最后通过显示设置栅栏使所有进程同步。  创建nbody目录，用于存放并行程序代码，利用mpicc工具进行编译，并使用mpirun -np x -f /home/zhangsan/nbody/config /home/zhangsan/nbody/nbody xxxx xxxx命令运行，分别设置进程数、数据规模、每个进程实际处理粒子数。  记录数据规模为6000时，单机、双机和三机随每机进程数变化的运行时间，以及每机1个进程时随数据规模变化的运行时间。根据记录的数据计算加速比与效率并绘图，其中加速比（T1表示最优串行算法在单处理器上的运行时间，Tp表示p个处理器的计算时间），效率（P为处理器数量）。  **2.3利用MPI对素数计算程序进行并行优化**  统计2-N之间素数的个数，N为用户输入。  将整数分解为P个子列表，如果有P个进程，那么只需给每个进程一个子列表进行计算，每个进程统计所负责子列表的素数个数并将其传输到进程0，进程0可以计算总数并进行输出。并行代码如下：   1. #include <stdlib.h> 2. #include <stdio.h> 3. #include <time.h> 4. #include <mpi.h> 6. *// 计算并返回当前进程负责范围内的质数数量* 7. int prime\_part(int id, int p, int n); 9. int main(int argc, char \*argv[]) { 10. int id, p, total, total\_part, n; 11. double start\_time, end\_time; 13. *// 初始化MPI环境* 14. MPI\_Init(&argc, &argv); 16. *// 获取进程总数和当前进程的编号* 17. MPI\_Comm\_size(MPI\_COMM\_WORLD, &p); 18. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id); 20. *// 检查命令行参数，确保提供了N的值* 21. if (argc < 2) { 22. if (id == 0) { 23. *// 如果未提供N的值，则打印使用方法* 24. printf("Usage: %s <N>\n", argv[0]); 25. } 26. *// 结束MPI环境并退出程序* 27. MPI\_Finalize(); 28. return 1; 29. } 31. *// 将命令行参数转换为整数* 32. n = atoi(argv[1]); 34. *// 开始计时* 35. start\_time = MPI\_Wtime(); 37. *// 计算当前进程负责范围内的质数数量* 38. total\_part = prime\_part(id, p, n); 40. *// 将所有进程的total\_part值汇总到进程0* 41. MPI\_Reduce(&total\_part, &total, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD); 43. *// 结束计时* 44. end\_time = MPI\_Wtime(); 46. *// 只有进程0打印结果* 47. if (id == 0) { 48. printf("\n"); 49. *// 打印2到n之间的质数总数和执行时间* 50. printf("     Between 2 and %d, there are %d primes\n", n, total); 51. printf("Execution Time: %f seconds\n", end\_time - start\_time); 52. } 54. *// 结束MPI环境* 55. MPI\_Finalize(); 56. return 0; 57. } 59. *// 计算并返回给定范围内的质数数量* 60. *// id: 当前进程的编号* 61. *// p: 进程总数* 62. *// n: 搜索质数的上限* 63. int prime\_part(int id, int p, int n) { 64. int i, j, prime, total\_part = 0; 66. *// 遍历当前进程负责的数字范围* 67. for (i = 2 + id; i <= n; i += p) { 68. prime = 1; *// 假设当前数字是质数* 70. *// 检查从2到sqrt(i)的因子* 71. for (j = 2; j \* j <= i; j++) { 72. if (i % j == 0) { 73. prime = 0; *// 发现因子，当前数字不是质数* 74. break; 75. } 76. } 77. *// 如果当前数字是质数且大于1* 78. if (prime && i > 1) { 79. total\_part++; *// 增加质数计数* 80. } 81. } 83. *// 返回当前进程找到的质数数量* 84. return total\_part; 85. }   记录N在不同取值下，随每机进程数变化的运行时间，计算程序加速比。  当前代码存在范围划分不均、素数检验耗时不均、静态调度导致节点闲置的问题，导致程序负载不均衡，可以采用动态调度策略，让空闲的进程获取新的任务；将整个范围划分为更小的块，并让多个进程分配这些块；改进素数检验部分的效率，如使用埃拉托斯特尼筛法等。改进后的并行代码如下：   1. #include <stdlib.h> 2. #include <stdio.h> 3. #include <time.h> 4. #include <mpi.h> 6. *// 判断一个数是否为质数* 7. int is\_prime(int number); 8. *// 计算并返回给定范围内的质数数量* 9. int prime\_part(int n, int start, int end); 11. int main(int argc, char \*argv[]) { 12. int id, p, total, total\_part, n; 13. double start\_time, end\_time; 15. *// 初始化MPI环境* 16. MPI\_Init(&argc, &argv); 18. *// 获取进程总数和当前进程的编号* 19. MPI\_Comm\_size(MPI\_COMM\_WORLD, &p); 20. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id); 22. *// 检查命令行参数，确保提供了N的值* 23. if (argc < 2) { 24. if (id == 0) { 25. *// 如果未提供N的值，则打印使用方法* 26. printf("Usage: %s <N>\n", argv[0]); 27. } 28. *// 结束MPI环境并退出程序* 29. MPI\_Finalize(); 30. return 1; 31. } 33. *// 将命令行参数转换为整数，表示搜索质数的上限* 34. n = atoi(argv[1]); 36. *// 开始计时* 37. start\_time = MPI\_Wtime(); 39. total\_part = 0; *// 初始化当前进程找到的质数数量为0* 40. int chunk\_size = 100; *// 设置每个进程处理的数字块大小* 42. *// 遍历当前进程负责的数字范围* 43. for (int start = 2 + id \* chunk\_size; start <= n; start += chunk\_size \* p) { 44. int end = start + chunk\_size - 1; *// 计算当前块的结束位置* 45. if (end > n) end = n; *// 如果结束位置超出n，则修正为n* 47. *// 计算当前块内的质数数量，并累加到total\_part* 48. total\_part += prime\_part(n, start, end); 49. } 51. *// 将所有进程的total\_part值汇总到进程0* 52. MPI\_Reduce(&total\_part, &total, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD); 54. *// 结束计时* 55. end\_time = MPI\_Wtime(); 57. *// 只有进程0打印结果* 58. if (id == 0) { 59. printf("\n"); 60. *// 打印2到n之间的质数总数和执行时间* 61. printf("     Between 2 and %d, there are %d primes\n", n, total); 62. printf("Execution Time: %f seconds\n", end\_time - start\_time); 63. } 65. *// 结束MPI环境* 66. MPI\_Finalize(); 67. return 0; 68. } 70. *// 计算并返回给定范围内的质数数量* 71. int prime\_part(int n, int start, int end) { 72. int i, total\_part = 0; 74. *// 遍历给定范围，判断每个数是否为质数* 75. for (i = start; i <= end && i <= n; i++) { 76. if (is\_prime(i)) { 77. total\_part++; *// 如果是质数，则增加计数* 78. } 79. } 81. return total\_part; *// 返回找到的质数数量* 82. } 84. *// 判断一个数是否为质数* 85. int is\_prime(int number) { 86. if (number < 2) return 0; *// 小于2的数不是质数* 87. *// 检查从2到sqrt(number)的因子* 88. for (int j = 2; j \* j <= number; j++) { 89. if (number % j == 0) return 0; *// 如果发现因子，则不是质数* 90. } 91. return 1; *// 如果没有发现因子，则是质数* 92. } | | | | | | | | | | | |
| **三、实验结果** | | | | | | | | | | | |
| **3.1 N体问题MPI并行程序剖析与性能分析**  实验一：单机上，数据规模为6000时，随每机进程数变化：  运行时间   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 进程数 | 1 | 2 | 3 | 4 | | 时间/s | 7.544928 | 3.785535 | 4.778441 | 4.248555 |     加速比（Sp）统计表   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 进程数 | 1 | 2 | 3 | 4 | | 加速比 | 1 | 1.99 | 1.58 | 1.78 |     效率（Ep）统计表   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 进程数 | 1 | 2 | 3 | 4 | | 加速比 | 1 | 0.995 | 0.526 | 0.445 |     实验二：相同数据规模为6000，随每机进程数变化：  运行时间   |  |  |  |  | | --- | --- | --- | --- | | 每机进程数 | 单机 | 双机 | 三机 | | 1 | 7.544928 | 3.777694 | 2.532161 | | 2 | 3.785535 | 1.917645 | 1.288663 | | 3 | 4.778441 | 2.982949 | 2.439660 | | 4 | 4.248555 | 2.951662 | 2.794911 |     加速比（Sp）统计表   |  |  |  |  | | --- | --- | --- | --- | | 每机进程数 | 单机 | 双机 | 三机 | | 1 | 1.00 | 2.00 | 2.98 | | 2 | 1.99 | 3.93 | 5.85 | | 3 | 1.58 | 2.53 | 3.09 | | 4 | 1.78 | 2.56 | 2.70 |     效率（Ep）统计表   |  |  |  |  | | --- | --- | --- | --- | | 每机进程数 | 单机 | 双机 | 三机 | | 1 | 1.000 | 1.000 | 0.993 | | 2 | 0.995 | 0.983 | 0.975 | | 3 | 0.527 | 0.422 | 0.343 | | 4 | 0.445 | 0.320 | 0.225 |     实验三：每机1个进程，随数据规模变化：  运行时间   |  |  |  |  | | --- | --- | --- | --- | | 粒子数n | 单机 | 双机 | 三机 | | 150 | 0.004836 | 0.004780 | 0.005635 | | 300 | 0.019018 | 0.015974 | 0.017812 | | 600 | 0.075543 | 0.042229 | 0.033772 | | 1200 | 0.302773 | 0.153546 | 0.169421 | | 2400 | 1.210833 | 0.606645 | 0.411603 | | 4800 | 4.826543 | 2.421853 | 1.617720 | | 9600 | 19.303552 | 9.636801 | 6.435894 |     加速比（Sp）统计表   |  |  |  |  | | --- | --- | --- | --- | | 粒子数n | 单机 | 双机 | 三机 | | 150 | 1 | 1.01 | 0.86 | | 300 | 1 | 1.19 | 1.07 | | 600 | 1 | 1.79 | 2.24 | | 1200 | 1 | 1.97 | 1.79 | | 2400 | 1 | 2.00 | 2.94 | | 4800 | 1 | 1.99 | 2.98 | | 9600 | 1 | 2.00 | 3.00 |     效率（Ep）统计表   |  |  |  |  | | --- | --- | --- | --- | | 粒子数n | 单机 | 双机 | 三机 | | 150 | 1 | 0.505 | 0.287 | | 300 | 1 | 0.595 | 0.357 | | 600 | 1 | 0.895 | 0.747 | | 1200 | 1 | 0.985 | 0.597 | | 2400 | 1 | 1.000 | 0.980 | | 4800 | 1 | 0.995 | 0.993 | | 9600 | 1 | 1.000 | 1.000 |     **3.2利用MPI对素数计算程序进行并行优化**  3.2.1改进前  N不同取值，随每机进程数变化：  运行时间   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 进程数  N | 1 | 2 | 4 | 6 | | 100000 | 0.006645 | 0.006599 | 0.003337 | 0.001397 | | 200000 | 0.017429 | 0.017339 | 0.008742 | 0.006643 | | 400000 | 0.046050 | 0.045667 | 0.028934 | 0.017046 | | 800000 | 0.122414 | 0.121728 | 0.064193 | 0.059328 |      |  |  |  |  |  | | --- | --- | --- | --- | --- | | 进程数  N | 1 | 2 | 4 | 6 | | 100000 | 1 | 1.01 | 1.99 | 4.76 | | 200000 | 1 | 1.01 | 1.99 | 2.62 | | 400000 | 1 | 1.01 | 1.59 | 2.70 | | 800000 | 1 | 1.01 | 1.91 | 2.06 |   加速比（Sp）统计表    3.2.2改进后  N不同取值，随每机进程数变化：  运行时间     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 进程数  N | 1 | 2 | 4 | 6 | | 100000 | 0.006770 | 0.003396 | 0.002044 | 0.001378 | | 200000 | 0.017690 | 0.008858 | 0.004818 | 0.003285 | | 400000 | 0.046428 | 0.023289 | 0.016444 | 0.014546 | | 800000 | 0.123932 | 0.062459 | 0.031043 | 0.023718 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 进程数  N | 1 | 2 | 4 | 6 | | 100000 | 1 | 1.99 | 3.31 | 4.91 | | 200000 | 1 | 2.00 | 3.67 | 5.39 | | 400000 | 1 | 1.99 | 2.82 | 3.19 | | 800000 | 1 | 1.98 | 3.99 | 5.23 |   加速比（Sp）统计表    **运行结果截图：** | | | | | | | | | | | |
| **四、思考题** | | | | | | | | | | | |
| 思考题1：从算法层面上分析素数计算程序负载不均衡的原因，并提出改进方法。  （1）原因：范围划分不均、素数检验耗时不均、静态调度导致节点闲置  （2）改进方法：采用动态调度策略，让空闲的进程获取新的任务；将整个范围划分为更小的块，并让多个进程分配这些块；改进素数检验部分的效率，如使用埃拉托斯特尼筛法等。  思考题2：根据该实验，简述并行程序编程和优化的步骤。  （1）确定并行化部分，将大问题分解为多个小问题；  （2）选择并行模型，使用已有的并行库或框架（如MPI、OpenMP等）；  （3）修改并行部分的代码，注意线程安全和数据共享的问题，避免竞争或死锁；  （4）对负载不均衡等性能瓶颈进行优化改进。 | | | | | | | | | | | |
| **五、实验心得体会** | | | | | | | | | | | |
| 1.学会了利用华为云服务器搭建并行计算的集群环境，了解了基于华为云的MPI并行程序设计流程。  2.通过N体问题的求解，掌握了MPI程序设计的基本编写、编译与运行方法，学会使用加速比、效率等性能评估方法。  3.学会了素数计算程序的并行程序编写，以及针对负载不均衡问题的改进。  4.对并行程序的设计编程和优化有了更深入理解。 | | | | | | | | | | | |
| 指导教师评语：  日期： | | | | | | | | | | | |