**高性能并行计算第8次作业**

姓名：代宏刚 学号：2020317110061

**一、寻找最大数**

**代码地址：**

见附录

**实验结果：**

表1 不同节点数下程序执行时间及结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 节点数  实验结果 | 串行 | 2 | 4 | MPI(4)+OMP(2) |
| 第1次 | 3.209634s | 1.719049s | 0.847344s | 0.545014s |
| 第2次 | 3.192576s | 1.591727s | 0.988776s | 0.562011s |
| 第3次 | 3.209746s | 1.620996s | 0.796684s | 0.557982s |
| 第4次 | 3.200125s | 1.621465s | 0.785223s | 0.564948s |
| 第5次 | 3.176172s | 1.618814s | 0.785241s | 0.580396s |
| 平均值 | 3.197651s | 1.63441s | 0.840654s | 0.56207s |
| 最大数字 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| 最大值位置 | 13068230 | 13068230 | 13068230 | 13068230 |

图1 不同节点数下程序执行时间

图2 不同节点数下程序加速比

图3 不同节点数下程序的效率

结果分析：程序随机生成1000000000个0-1的随机浮点数，为比较程序正确性，在每个程序中都以1作为seed生成伪随机数。并且生成随机数的时间不计入程序执行之间内。

加速比=串行执行时间/并行执行时间

并行效率=加速比/节点数。

随着节点数增加，程序执行时间减少。在没有引入omp线程时，mpi的并行效率都在0.95以上，但加入omp双线程并行后，并行效率降到了0.71。在omp线程并行中，采用的是for循环并行，在线程中的并行效率比较低。

**二、Pi-MPI+OpenMP混合编程**

**代码地址：**

见附录

**实验结果：**

表1 不同进程数下程序执行时间

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 进程数  实验结果 | 串行 | 2 | 4 | 6 | 8 |
| 第1次 | 17.1 | 8.42 | 2.12 | 1.43 | 1.10 |
| 第2次 | 16.8 | 8.43 | 2.12 | 1.43 | 1.07 |
| 第3次 | 16.81 | 8.42 | 2.12 | 1.43 | 1.08 |
| 第4次 | 16.8 | 8.41 | 2.12 | 1.47 | 1.10 |
| 第5次 | 16.8 | 8.43 | 2.13 | 1.41 | 1.08 |
| 平均值 | 16.862 | 8.422 | 2.122 | 1.434 | 1.086 |

图1 不同进程数下程序加速比

图2: 不同进程数下程序效率

结果分析：

不同进程数下程序执行时间以5次执行的平均值表示。串行执行时间为程序在单进程单线程下执行的时间，其它进程数为在对应进程数下程序以双线程执行的时间。

加速比=串行执行时间/并行执行时间

并行效率=加速比/节点数。

从图2可看出，随着进程数增加，程序加速比越来越大。但在图2中。当进程数为2(线程数为2)时，程序效率几乎降低了一半，当进程数为4,6,8时，程序并行效率依然很高。在程序执行时检测cpu使用情况发现，在进程数为2时，cpu使用约为100%，在进程数为4,6,8时，cpu使用约为200%，说明进程数为2时，两个线程并没有并行起来。当加上—bind-to core 参数执行程序时，无论进程数为多少，线程之间始终不能并行。

## 附录

## 一、寻找最大数

①串行程序

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include "omp.h"
4. #include "time.h"
5. #include "crt\_rand.h"
6. //定义结构体
7. **typedef** **struct**{
8. **float** value;
9. **int** index;
10. } mess;
11. **int** main(**int** argc, **char**\* argv[]){
12. **int** i,j,N;
13. **double** t1, t2;
14. N = atoi(argv[1]);
15. **float**\* a;
16. a = crtarr(N,1); //生成N个0-1的随机数
17. mess\* arr = NULL;
18. arr = (mess\*)malloc(**sizeof**(mess)\*N);
19. **if**(arr==NULL){
20. printf("内存分配失败\n");
21. exit(-1);
22. }
23. **for**(i=0;i<N;i++){
24. arr[i].value = a[i];
25. arr[i].index = i;
26. } //将生成的N个随机数依次赋值给arr数组
27. free(a);
28. a = NULL;
29. mess max, min, out;
30. max = min=arr[0];
31. t1 = omp\_get\_wtime();
32. **for**(i=0;i<N;i++){
33. **if**(max.value<arr[i].value)
34. max = arr[i];
35. }
36. free(arr);
37. arr = NULL;
38. t2 = omp\_get\_wtime();
39. printf("maxindex=%d,maxvalue=%f,time=%f(s)\n",max.index,max.value,t2-t1);
40. **return** 0;
41. }

②mpi并行程序

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include "time.h"
4. #include "mpi.h"
5. #include "crt\_rand.h"
6. //定义结构体
7. **typedef** **struct**{
8. **float** value;
9. **int** index;
10. } mess;
11. **int** main(**int** argc, **char**\* argv[]){
12. **int** i,j,N;
13. **double** t1, t2;
14. N = atoi(argv[1]);
15. **float**\* a;
16. a = crtarr(N,1);
17. **int** world\_rank, world\_size;
18. mess\* arr = NULL;
19. arr = (mess\*)malloc(**sizeof**(mess)\*N);
20. **if**(arr==NULL){
21. printf("内存分配失败\n");
22. exit(-1);
23. }
24. **for**(i=0;i<N;i++){
25. arr[i].value = a[i];
26. arr[i].index = i;
27. }
28. free(a);
29. a = NULL;
30. MPI\_Init(&argc, &argv);
31. MPI\_Comm\_rank(MPI\_COMM\_WORLD,&world\_rank);
32. MPI\_Comm\_size(MPI\_COMM\_WORLD,&world\_size);
33. **int** step = N/world\_size;
34. mess max, out;
35. max = arr[world\_rank\*step];
36. t1 = MPI\_Wtime();
37. **if**(world\_rank!=world\_size-1){
38. **for**(i=world\_rank\*step;i<(world\_rank+1)\*step;i++){
39. **if**(max.value<arr[i].value)
40. max = arr[i];
41. }
42. }
43. **else**{
44. **for**(i=world\_rank\*step;i<N;i++){
45. **if**(max.value<arr[i].value)
46. max = arr[i];
47. }
48. }
49. free(arr);
50. arr = NULL;
51. MPI\_Barrier(MPI\_COMM\_WORLD);
52. MPI\_Reduce(&max,&out,1,MPI\_FLOAT\_INT,MPI\_MAXLOC,0,MPI\_COMM\_WORLD);
53. t2 = MPI\_Wtime();
54. **if**(world\_rank==0)
55. printf("maxrank=%d,maxindex=%d,maxvalue=%f,time=%f(s)\n",out.index/N,out.index%N,out.value,t2-t1);
57. **return** MPI\_Finalize();
58. }

③.mpi+omp混合并行

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include "omp.h"
4. #include "time.h"
5. #include "mpi.h"
6. #include "crt\_rand.h"
7. **typedef** **struct**{
8. **float** value;
9. **int** index;
10. } mess;
11. **int** main(**int** argc, **char**\* argv[]){
12. **int** i,j,N;
13. **double** t1, t2;
14. N = atoi(argv[1]);
15. **float**\* a;
16. a = crtarr(N,1);
17. **int** world\_rank, world\_size;
18. mess\* arr = NULL;
19. arr = (mess\*)malloc(**sizeof**(mess)\*N);
20. **if**(arr==NULL){
21. printf("内存分配失败\n");
22. exit(-1);
23. }
24. **for**(i=0;i<N;i++){
25. arr[i].value = a[i];
26. arr[i].index = i;
27. }
28. free(a);
29. a = NULL;
30. **int** provided;
31. MPI\_Init\_thread(&argc, &argv, MPI\_THREAD\_FUNNELED,&provided);
32. MPI\_Comm\_rank(MPI\_COMM\_WORLD,&world\_rank);
33. MPI\_Comm\_size(MPI\_COMM\_WORLD,&world\_size);
34. **int** step = N/world\_size;
35. mess max, out;
36. max = arr[world\_rank\*step];
37. t1 = omp\_get\_wtime();
38. **if**(world\_rank!=world\_size-1){
39. #pragma omp parallel for
40. **for**(i=world\_rank\*step;i<(world\_rank+1)\*step;i++){
41. **if**(max.value<arr[i].value)
42. max = arr[i];
43. }
44. }
45. **else**{
46. #pragma omp parallel for
47. **for**(i=world\_rank\*step;i<N;i++){
48. **if**(max.value<arr[i].value)
49. max = arr[i];
50. }
52. }
53. free(arr);
54. arr = NULL;
55. MPI\_Barrier(MPI\_COMM\_WORLD);
56. MPI\_Reduce(&max,&out,1,MPI\_FLOAT\_INT,MPI\_MAXLOC,0,MPI\_COMM\_WORLD);
57. t2 = omp\_get\_wtime();
58. **if**(world\_rank==0)
59. printf("maxrank=%d,maxindex=%d,maxvalue=%f,time=%f(s)\n",out.index/N,out.index%N,out.value,t2-t1);
60. **return** MPI\_Finalize();
61. }

**二、Pi-MPI+OpenMP混合编程**

1. #include "mpi.h"
2. #include "omp.h"
3. #include <math.h>
4. #include <stdio.h>
5. #define N 1000000000
6. **int** main(**int** argc, **char**\* argv[]){
7. **int** rank, nproc;
8. **int** i, low, up;
9. **double** local = 0.0, pi, w, temp, t0, t1, t2;
10. **int** provided;
11. MPI\_Status status;
12. MPI\_Init\_thread(&argc, &argv, MPI\_THREAD\_FUNNELED, &provided);
13. MPI\_Comm\_size(MPI\_COMM\_WORLD, &nproc);
14. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
15. t0 = MPI\_Wtime();
16. w = 1.0/N ; low = rank\*(N/nproc); up = low +N/nproc - 1;
17. #pragma omp parallel for reduction(+:local) private(temp,i)
18. **for** (i=low;i<up;i++){
19. temp = (i+0.5) \* w;
20. local = local + 4.0/(1.0+temp\*temp);
21. }
22. MPI\_Reduce(&local,&pi,1,MPI\_DOUBLE,MPI\_SUM,0,MPI\_COMM\_WORLD);
23. **if**(rank==0) printf("pi=%.20f\n",pi\*w);
24. t1 = MPI\_Wtime();
25. **if**(rank==0) printf("time used = %.2f\n", t1-t0);
26. **return** MPI\_Finalize();
27. }