**高性能并行计算第2次作业**

姓名：代宏刚 学号：2020317110061

**代码地址：**

**C语言：**

1. #include <stdio.h>
2. #include <malloc.h>
3. #include <time.h>
4. #include <stdlib.h>
5. //打印矩阵
6. void print\_arr(int\*\* arr, int N){
7. int row,col;
8. **for**(row=0;row<N;row++){
9. **for**(col=0;col<N;col++){
10. printf("%d\t",arr[row][col]);
11. }
12. printf("\n");
13. }
14. printf("\n");
15. }
16. int main(){
17. //利用时间随机作种
18. srand((unsigned) time(NULL));
19. int N,row,col,k,temp;
20. int \*\*arr\_1, \*\*arr\_2,\*\*result = NULL;
21. clock\_t start, end;
22. printf("输入数组维度：");
23. scanf("%d",&N);
24. //分配矩阵内存空间
25. start = clock();
26. arr\_1 = (int\*\*)malloc(sizeof(int\*)\*N);
27. arr\_2 = (int\*\*)malloc(sizeof(int\*)\*N);
28. result = (int\*\*)malloc(sizeof(int\*)\*N);
29. **if**(arr\_1==NULL||arr\_2==NULL||result==NULL){
30. printf("内存分配失败");
31. exit(-1);
32. }
33. **for**(k=0;k<N;k++){
34. arr\_1[k] = (int\*)malloc(sizeof(int)\*N);
35. arr\_2[k] = (int\*)malloc(sizeof(int)\*N);
36. result[k] = (int\*)malloc(sizeof(int)\*N);
37. **if**(arr\_1[k]==NULL||arr\_2[k]==NULL||result[k]==NULL){
38. printf("内存分配失败");
39. exit(-1);
40. }
41. }
42. //数组随机赋1-100初值
43. **for**(row=0;row<N;row++){
44. **for**(col=0;col<N;col++){
45. arr\_1[row][col] = rand() % 100;
46. arr\_2[row][col] = rand() % 100;
47. }
48. }
49. //打印输入矩阵
50. print\_arr(arr\_1,N);
51. print\_arr(arr\_2,N);
53. //矩阵乘法计算
54. **for**(row=0;row<N;row++){
55. **for**(col=0;col<N;col++){
56. temp = 0;
57. **for**(k=0;k<N;k++){
58. temp += arr\_1[row][k] \* arr\_2[k][col];
59. }
60. result[row][col] = temp;
61. }
62. printf("\n");
63. }
64. //打印输出矩阵
65. print\_arr(result,N);
66. end = clock();
67. printf("clock=%.3f",(double)(end-start)/CLOCKS\_PER\_SEC);
68. free(arr\_1);
69. free(arr\_2);
70. free(result);
71. arr\_1 = NULL;
72. arr\_2 = NULL;
73. result  = NULL;
74. **return** 0;
75. }

**Python:（使用numpy）**

1. **from** numpy **import** matrix
2. **import** random
3. N = int(input("input scale N:"))  # 输入矩阵大小
4. #  随机生成两个N\*N的矩阵
5. li\_1 = [[random.randint(1, 99) **for** i **in** range(N)] **for** j **in** range(N)]
6. li\_2 = [[random.randint(1, 99) **for** i **in** range(N)] **for** j **in** range(N)]
7. matrix\_1 = matrix(li\_1)
8. matrix\_2 = matrix(li\_2)
9. #  打印矩阵
10. **print**(matrix\_1)
11. **print**(matrix\_2)
12. result = matrix\_1 \* matrix\_2
13. **print**(result)

**Python(不使用numpy)**

1. **import** random
2. N = int(input("input scale N:"))
3. # 随机生成乘数矩阵
4. arr\_1 = [[random.randint(1,99) **for** i **in** range(N)] **for** j **in** range(N)]
5. arr\_2 = [[random.randint(1,99) **for** i **in** range(N)] **for** j **in** range(N)]
6. result = [[ ] **for** i **in** range(N)]
7. **print**(arr\_1)
8. **print**(arr\_2)
9. **for** i **in** range(N):
10. **for** j **in** range(N):
11. tmp = 0
12. **for** k **in** range(N):
13. tmp += arr\_1[i][k] \* arr\_2[k][j]
14. result[i].append(tmp)
15. **print**(result)

**实验结果：**

表1：C程序执行时间

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 10 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 |
| time(s) | 0.001 | 0.026 | 0.108 | 0.276 | 0.576 | 1.080 | 1.829 | 2.546 | 3.799 | 5.219 | 7.109 |
| clock(s) | 0.00 | 0.01 | 0.10 | 0.26 | 0.56 | 1.07 | 1.81 | 2.54 | 3.79 | 5.21 | 7.10 |

结果分析：

**N**:矩阵维数，在程序执行时由键盘输入，N的取值从10到1000，如表所示

**time**:利用linux的time命令，计算程序运行时间，因为程序中存在scanf语句，需要获取键盘输入值作为N值，所以time取值为sys时间和user时间之和。

**clock**:利用c语言的clock库函数，计算出的程序执行时间，因为clock函数的精度问题，在N值小于100时，无法计算出程序执行时间。

在程序中，根据输入的N值分别分配两个乘数矩阵和结果矩阵的内存空间，利用random随机数生成乘数矩阵的数值。从结果可看出，程序执行时间的增长速率要快于N值增长的速率，当N值增加时，需要生成的矩阵元素个数以期平方倍增加，另外，在计算矩阵乘法时，用到了三重的嵌套for循环结构，导致程序执行时间增长的速率比N值快6~8倍。

表2：Python程序执行时间（使用numpy）

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 10 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 |
| time(s) | 0.137 | 0.157 | 0.232 | 0.371 | 0.625 | 0.919 | 1.307 | 1.766 | 2.426 | 3.007 | 3.740 |

Python程序执行时间(不使用numpy)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | 10 | 50 | 100 | 150 | 200 | 250 |
| time(s) | 0.031 | 0.071 | 0.306 | 0.964 | 2.128 | 4.379 |

结果分析:

**N**:矩阵维数，在程序执行时由键盘输入，N的取值从10到1000，如表所示

**time**:利用linux的time命令，计算程序运行时间，因为程序中存在scanf语句，获取键盘输入值作为N值，所有time取值为sys时间和user时间之和。

在Python中计算矩阵乘法时，用到了numpy中的matrix方法。

当N小于500时，用c语言编写的程序执行效率要高于python编写的程序。当N>=500时，Python程序的执行效率要高于c程序。当N值越大时，Python的优势越显著。可能是因为numpy的底层为c语言程序，且对矩阵乘法的算法做了优化。

若使用纯Python计算矩阵乘法，效率非常低，当值增到250时，程序执行时间就已经高达4.3秒。