

Lab 1

721030290001

张艺珩

计算机科学

1a)



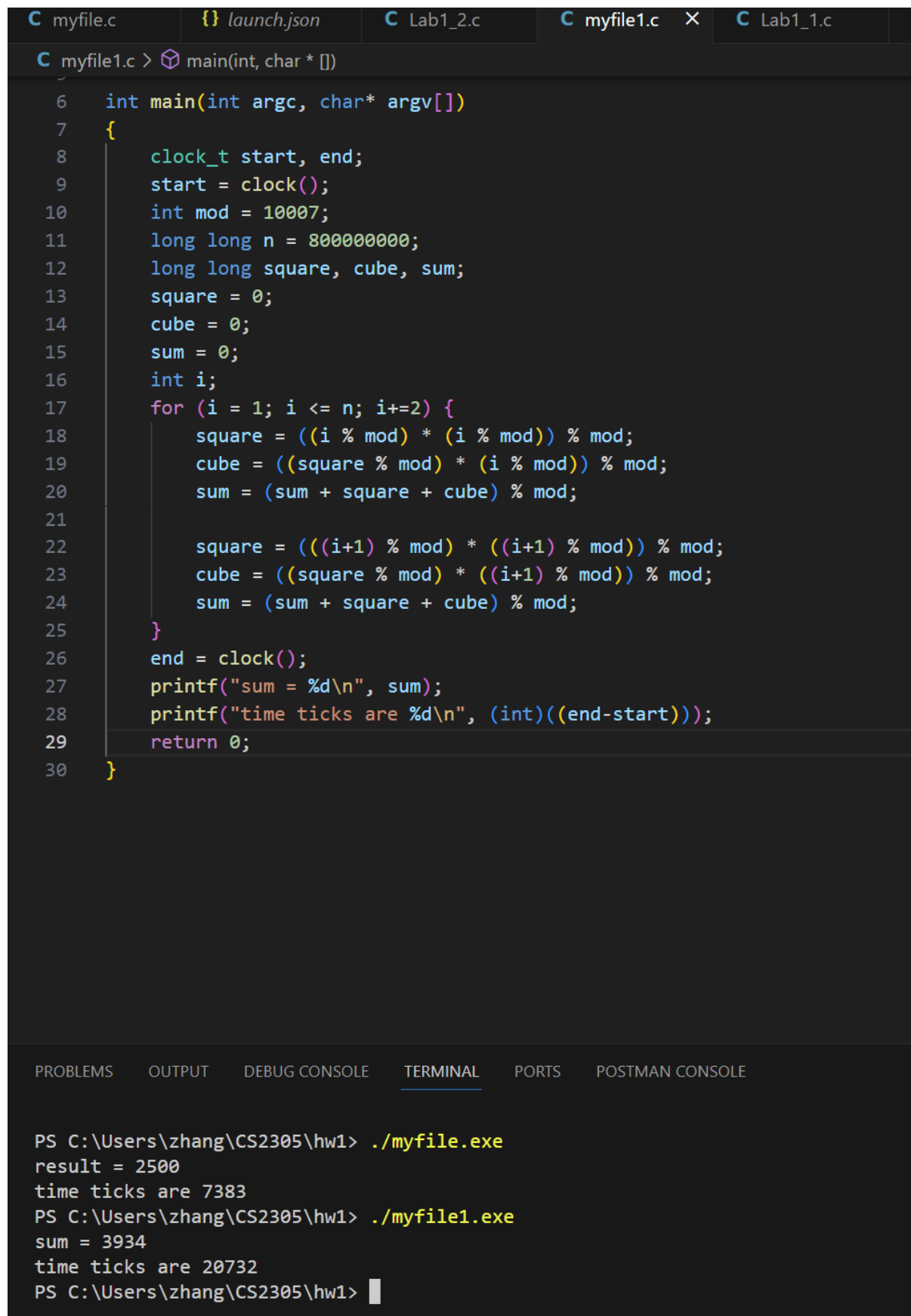
The image shows a code editor with a dark theme. At the top, there are tabs for 'myfile.c', 'launch.json', 'Lab1_2.c', 'myfile1.c', and 'Lab1_1.c'. The 'myfile.c' tab is active, showing a C program. The program includes headers for stdio.h, stdlib.h, omp.h, and time.h. It defines a main function that takes argc and argv as arguments. Inside the main function, it declares variables for clock_t (start, end), int (mod), long long (n, first, second, result), and int (i). It initializes start to clock(), mod to 10007, n to 800000001, first to 0, second to 1, and result to 0. It then enters a for loop from i = 2 to n, where it calculates result = (first + second) % mod, updates first = second, and updates second = result. After the loop, it sets end = clock(), prints the result and the time ticks, and returns 0.

```
C myfile.c X {} launch.json C Lab1_2.c C myfile1.c C Lab1_1.c
C myfile.c > main(int, char * [])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4  #include <time.h>
5
6  int main(int argc, char* argv[])
7  {
8
9      clock_t start, end;
10     start = clock();
11     int mod = 10007;
12     long long n = 800000001;
13     long long first, second, result;
14     first = 0;
15     second = 1;
16     result = 0;
17
18     int i;
19     for (i = 2; i <= n; ++i) {
20         result = (first + second) % mod;
21         first = second;
22         second = result;
23     }
24     end = clock();
25     printf("result = %d\n", result);
26     printf("time ticks are %d\n", (int)((end-start)));
27     return 0;
28 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
PS C:\Users\zhang\CS2305\hw1> ./myfile.exe
result = 2500
time ticks are 7383
PS C:\Users\zhang\CS2305\hw1> 
```

1b)



The image shows a Visual Studio Code editor window with a C program named `myfile1.c` open. The program calculates the sum of squares and cubes of odd numbers from 1 to 8000000000, modulo 10007, and measures the execution time. The terminal at the bottom shows the execution of `myfile.exe` and `myfile1.exe`, displaying the results of the calculations and the time taken.

```
C myfile.c {} launch.json C Lab1_2.c C myfile1.c X C Lab1_1.c
C myfile1.c > main(int, char * [])

6  int main(int argc, char* argv[])
7  {
8      clock_t start, end;
9      start = clock();
10     int mod = 10007;
11     long long n = 8000000000;
12     long long square, cube, sum;
13     square = 0;
14     cube = 0;
15     sum = 0;
16     int i;
17     for (i = 1; i <= n; i+=2) {
18         square = ((i % mod) * (i % mod)) % mod;
19         cube = ((square % mod) * (i % mod)) % mod;
20         sum = (sum + square + cube) % mod;
21
22         square = (((i+1) % mod) * ((i+1) % mod)) % mod;
23         cube = ((square % mod) * ((i+1) % mod)) % mod;
24         sum = (sum + square + cube) % mod;
25     }
26     end = clock();
27     printf("sum = %d\n", sum);
28     printf("time ticks are %d\n", (int)((end-start)));
29     return 0;
30 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
PS C:\Users\zhang\CS2305\hw1> ./myfile.exe
result = 2500
time ticks are 7383
PS C:\Users\zhang\CS2305\hw1> ./myfile1.exe
sum = 3934
time ticks are 20732
PS C:\Users\zhang\CS2305\hw1> 
```

2) The 2nd code block is more suitable for multithreading because in every iteration, the calculations are based on the current iteration number, therefore, the iterations can be assigned to different threads and the results wouldn't be affected. On the other hand, for the Fibonacci code, each iteration is dependent on the results from the previous iteration, multithreading the code will lead to it producing the wrong result.

3)

```
Lab1_1.c > main(int, char * [])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4  #include <time.h>
5
6  int main(int argc, char* argv[])
7  {
8      clock_t start, end;
9      start = clock();
10     int mod = 10007;
11     long long n = 800000000;
12     long long square, cube, sum, sum_square, sum_cube;
13     sum_square = 0;
14     sum_cube = 0;
15     square = 0;
16     cube = 0;
17     sum = 0;
18     int i;
19     #pragma omp parallel for num_threads(4) private(square, cube) reduction(+:sum_square, sum_cube)
20     for (i = 1; i <= n; ++i) {
21         square = ((i % mod) * (i % mod)) % mod;
22         cube = ((square % mod) * (i % mod)) % mod;
23         sum_square += square;
24         sum_cube += cube;
25     }
26     sum = (sum_square + sum_cube) % mod;
27     end = clock();
28     printf("sum = %d\n", sum);
29     printf("time ticks are %d\n", (int)((end-start)));
30     return 0;
31 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
PS C:\Users\zhang\CS2305\hw1> ./myfile1.exe
sum = 3934
time ticks are 20552
PS C:\Users\zhang\CS2305\hw1> ./Lab1_1.exe
sum = 3934
time ticks are 4890
PS C:\Users\zhang\CS2305\hw1> 
```

By implementing parallelism with OpenMP and utilizing 4 threads in my code, there is a significant performance improvement. The speedup achieved is approximately 4 times compared to the sequential version of the code. This improvement is evident from the reduced execution time, as indicated by the lower number of time ticks for the parallelized version (Lab1_1.exe) compared to the original version (myfile1.exe). The time ticks represent the CPU clock cycles consumed during code execution. The parallelized code demonstrates a notable reduction in time ticks, indicating that it completes the same amount of work in less time compared to the sequential code. The speedup of around 4 times, which is calculated by dividing the time ticks of the different code, highlights the effectiveness of utilizing multiple threads and parallelism in

improving performance. Overall, the introduction of parallelism using OpenMP, with its division of workload among multiple threads, has yielded a significant performance improvement, resulting in faster execution and a noticeable speedup.

4) It will be calculated in t_2 .

5)

```
C myfile.c  X  C Lab1_2.c  C myfile1.c  C Lab1_1.c
C myfile.c > main(int, char * [])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4  #include <time.h>
5
6  int main(int argc, char* argv[])
7  {
8
9      clock_t start, end;
10     start = clock();
11     int mod = 10007;
12     long long n = 800000001;
13     long long first, second, result;
14     first = 0;
15     second = 1;
16     result = 0;
17     int i;
18     for (i = 2; i <= n; ++i) {
19         result = (first + second) % mod;
20         first = second;
21         second = result;
22     }
23     end = clock();
24     printf("result = %d\n", result);
25     printf("time ticks are %d\n", (int)((end-start)));
26     return 0;
27 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
PS C:\Users\zhang\CS2305\hw1> ./Lab1_2.exe
result = 2500
time ticks are 7180
PS C:\Users\zhang\CS2305\hw1> ./Lab1_2.exe
result = 2500
time ticks are 7362
PS C:\Users\zhang\CS2305\hw1> ./Lab1_2.exe
result = 2500
time ticks are 7127
PS C:\Users\zhang\CS2305\hw1> ./Lab1_2.exe
result = 2500
time ticks are 7090
PS C:\Users\zhang\CS2305\hw1> ./Lab1_2.exe
result = 2500
time ticks are 7184
PS C:\Users\zhang\CS2305\hw1> 
```