**The title of the university: HDU-ITMO**

**Group member: Wang Xinyue, Fan Xulin, Shi Chiye**

**Laboratory research #3 ”Loop parallelization using**

**OpenMP technology”**

目录

[1 Processor description 3](#_Toc38322086)

[2 Description of the parallel library configuration features 4](#_Toc38322087)

[3 Full text of the resulting parallel program and the scripts 5](#_Toc38322088)

[4 Graphs of program execution time functions 7](#_Toc38322089)

[5 Detailed conclusions with an analysis of the graphs and the results obtained. 8](#_Toc38322090)

# 1 Processor description

Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz

Operating system: ubuntu18.04

GCC compiler used for experiments: gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)

bit depth: 64

number of cores: 4

RAM capacity: 8G

# 2 Description of the parallel library configuration features

**a. First of all, add OpenMP Directive to all for-loops in the program.** **Check all for-loops for internal dependencies on data between iterations. If dependencies are found, use the directive ” #pragma omp critical ” or ” #pragma omp atomic ” (if the operation is atomic), or the reduction parameter (preferably), or refuse to parallelize the loop at all to protect critical sections.**

**b. Second, Conduct experiments by measuring the parallel acceleration. Compare parallel acceleration graphs with Laboratory research #1 and #2. Conduct the experiments by adding the ”schedule” parameter and varying the schedule type in the experiments.** **Compare the parallel acceleration for different schedules with the results of all variables. Try different values of thread number to find the best.**

**c. Third, analyze the result and discuss.**

# 3 Full text of the resulting parallel program and the scripts

The resulting parallel program

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <math.h>
4. #include <sys/time.h>
5. int main(int argc, char \*argv[])
6. {
7. int i, N,j;
8. int M,thread;
9. struct timeval T1, T2;
10. long delta\_ms;
11. N = atoi(argv[1]);
12. thread = atoi(argv[2]);
13. M=5;
14. //N = 10000;
15. gettimeofday(&T1, NULL);
16. unsigned int seed = 0;
17. #pragma omp parallel for num\_threads(thread)
18. for (i = 0; i < 50; i++) /\* 50 experiments50 experiments \*/
19. {
20. double M1[N] ,M2[(int)(N/2)];
21. srand(i);
22. int A = 24;
23. #pragma omp parallel for private(j) shared(M1) schedule(guided,M)
24. for(j = 0; j < N; j++){
25. M1[j] = rand\_r(&seed)%A;
26. while(M1[j]==0.0){
27. M1[j] = rand\_r(&seed)%A;
28. }
29. }
30. #pragma omp parallel for private(j) shared(M2) schedule(guided,M)
31. for(j = 0; j < N/2; j++){
32. M2[j] = rand\_r(&seed)%(A\*10);
33. }
35. /\* Map Stage. Apply Hyperbolic cotangent of number’s root to each element in the M1 array \*/
36. #pragma omp parallel for private(j) shared(M1) schedule(guided,M)
37. for(j = 0; j < N; j++){
38. M1[j] = sqrt(1.0/tanh(M1[j]));
39. }
40. /\* in the M2 array, add each element in turn with the previous one ,and apply Sine modulus to the result of the addition\*/
41. double M2\_temp[(int)(N/2)+1];
42. M2\_temp[0] = 0;
43. #pragma omp parallel for private(j) shared(M2\_temp,M2) schedule(guided,M)
44. for(j = 1; j < N/2+1; j++){
45. M2\_temp[j] = M2[j-1];
46. }
47. #pragma omp parallel for private(j) shared(M2) schedule(guided,M)
48. for(j = 0; j < N/2; j++){
49. M2[j] = fabs(sin(M2\_temp[j]+M2\_temp[j+1]));
51. }
52. /\* Merge Stage. In arrays M1 and M2 apply Raising to a power to all elements in pairs with the same indexes in \*/
53. #pragma omp parallel for private(j) shared(M2,M1) schedule(guided,M)
54. for(j = 0; j < N/2; j++){
55. //#pragma omp atomic
56. M2[j] = pow(M1[j],M2[j]);
57. }
59. /\* Stupid sort \*/
60. int x = 0;
61. while (x < N/2)
62. {
63. if (x == 0 || M2[x - 1] <= M2[x])
64. x++;
65. else
66. {
67. double tmp = M2[x];
68. M2[x] = M2[x - 1];
69. M2[--x] = tmp;
70. }
71. }
72. /\* Reduce Stage. determining parity \*/
73. double min = A\*10;
74. #pragma omp parallel for private(j) shared(M2,min) schedule(guided,M)
75. for(j = 0; j < N/2; j++){/\* Find the minimum \*/
76. if(min > M2[j] && M2[j]!=0)
77. #pragma omp critical
78. min = M2[j];
79. //printf(" %f ",M2[j]);
80. }
81. //printf("min: %f\n",min);
82. double sum = 0;
83. #pragma omp parallel for private(j) reduction(+:sum) shared(M2) schedule(guided,M)
84. for(j = 0; j < N/2; j++){
85. int tmp = (int)(M2[j]/min);
86. if(tmp%2==0.0)
87. sum = sum + sin(M2[j]);
88. }
89. //printf("X: %f\n",sum);
90. }
92. gettimeofday(&T2, NULL);
93. /\* remember the current time T2 \*/
94. delta\_ms = 1000 \* (T2.tv\_sec - T1.tv\_sec) + (T2.tv\_usec - T1.tv\_usec) / 1000;
95. printf("\nN=%d. Milliseconds passed: %ld\n", N, delta\_ms); /\* T2 -T1 \*/
96. return 0;

}

# 4 Graphs of program execution time functions

**Because there is no significant difference between the results of different values of ‘schedule’. I did not draw the plot of it. I chose ‘guided=5’ as the result and the basement of the next step of the lab work. The results of this part are in the sheet “schedule” in “lab3.xlsx”.**

**As the plot thread number result shows, thread number equals 4 is the best result. It’s consistent of the number of the CPU cores.**

# 5 Detailed conclusions with an analysis of the graphs and the results obtained.

The value of schedule and the chunk-size did not influence the result. That reason may be that in every stage that used omp parallel, there are barely no difference when using different strategies of schedule.

From the result of plot compare of the results, we can infer that although with the efficiency with the opm method, 33% optimization can be achieved. But its still worese than the result of using Frame wave. The reason may be that in the sort stage, I failed to find a way to do the parallel with OpenMP. In sort stage, there are actually no optimization in my lab3 work. This failure can greatly influence the result.