

# High Performance Computing

## CS5013

### Lab 02

R Abinav  
ME23B1004

January 21, 2026

## 1 Implementation

### 1.1 Parallel Code Using Reduction Construct

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<omp.h>
4 #include<time.h>
5
6 #define n 100000000
7
8 int main(){
9     srand(time(NULL));
10    double sum = 0;
11
12    int threads[] = {1, 2, 4, 6, 8, 10, 12, 16, 20, 32, 64};
13    int num_tests = sizeof(threads)/sizeof(threads[0]);
14
15    double t_1 = -1;
16
17    double *numbers = (double*)malloc(n * sizeof(double));
18    for(int i=0; i<n; i++){
19        numbers[i] = (double)rand() * 1000000;
20    }
21
22    //warm up
23    printf("Warm up\n");
24    omp_set_num_threads(threads[0]);
25    #pragma omp parallel reduction(+:sum)
26    {
27        #pragma omp for
28        for(int i=0; i<n; i++){
29            sum += numbers[i];
30        }
31    }
```

```

31 }
32 printf("Warm up complete\n");
33
34 for(int i=0; i<num_tests; i++){
35     sum = 0;
36
37     omp_set_num_threads(threads[i]);
38     double s_time = omp_get_wtime();
39
40     #pragma omp parallel reduction(+:sum)
41     {
42         #pragma omp for
43         for(int j=0; j<n; j++){
44             sum += numbers[j];
45         }
46     }
47
48     printf("Sum: %lf\n", sum);
49
50     double e_time = omp_get_wtime();
51     double exec_time = e_time - s_time;
52     printf("%d thread(s) exec time: %lf\n", threads[i],
53            exec_time);
54
55     double speedup = -1.00;
56     if(i == 0){
57         t_1 = exec_time;
58         speedup = 1.00;
59         printf("The speedup (for %d thread(s)) is: %lf\n",
60                threads[i], speedup);
61     }else{
62         speedup = (double)t_1/exec_time;
63         printf("The speedup (for %d thread(s)) is: %lf\n",
64                threads[i], speedup);
65     }
66
67     double f = (threads[i] * (speedup - 1)) /
68                 ((threads[i] - 1) * speedup);
69     printf("Parallelisation factor (thread(s) = %d): %lf\n",
70            threads[i], f);
71
72     printf("\n\n");
73 }
74 free(numbers);
75 return 0;
76 }
```

Listing 1: OpenMP Parallel Sum with Reduction

## 1.2 Parallel Code Using Critical Section

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<omp.h>
4 #include<time.h>
5
6 #define n 100000000
7
8 int main(){
9     srand(time(NULL));
10
11     int threads[] = {1, 2, 4, 6, 8, 10, 12, 16, 20, 32, 64};
12     int num_tests = sizeof(threads)/sizeof(threads[0]);
13
14     double t_1 = -1;
15
16     double *numbers = (double*)malloc(n * sizeof(double));
17     for(int i=0; i<n; i++){
18         numbers[i] = (double)rand() * 1000000;
19     }
20
21 //warm up
22 printf("Warm up\n");
23 double sum_temp = 0;
24 omp_set_num_threads(threads[0]);
25 #pragma omp parallel
26 {
27     #pragma omp for
28     for(int i=0; i<n; i++){
29         sum_temp += numbers[i];
30     }
31 }
32 printf("Warm up complete\n");
33
34 for(int i=0; i<num_tests; i++){
35     double sum = 0;
36     omp_set_num_threads(threads[i]);
37     double s_time = omp_get_wtime();
38
39     #pragma omp parallel
40     {
41         #pragma omp for
42         for(int j=0; j<n; j++){
43             #pragma omp critical
44             {
45                 sum += numbers[j];
46             }
47         }
48     }
49 }
```

```

50     printf("Sum: %lf\n", sum);

51

52     double e_time = omp_get_wtime();
53     double exec_time = e_time - s_time;
54     printf("%d thread(s) exec time: %lf\n", threads[i],
55            exec_time);

56     double speedup = -1.00;
57     if(i == 0){
58         t_1 = exec_time;
59         speedup = 1.00;
60         printf("The speedup (for %d thread(s)) is: %lf\n",
61                threads[i], speedup);
62     }else{
63         speedup = (double)t_1/exec_time;
64         printf("The speedup (for %d thread(s)) is: %lf\n",
65                threads[i], speedup);
66     }

67     double f = (threads[i] * (speedup - 1)) /
68             ((threads[i] - 1) * speedup);
69     printf("Parallelisation factor (thread(s) = %d): %lf\n",
70            threads[i], f);

71     printf("\n\n");
72 }

73     free(numbers);
74     return 0;
75 }

```

Listing 2: OpenMP Parallel Sum with Critical Section

## 2 Results and Analysis

### 2.1 Performance Data - Reduction Construct

Table 1: Execution Time and Speedup with Reduction

Threads	Execution Time (s)	Speedup
1	0.348574	1.000000
2	0.187231	1.861731
4	0.096168	3.624639
6	0.088245	3.950057
8	0.056314	6.189830
10	0.063332	5.503914
12	0.058904	5.917702
16	0.060231	5.787323
20	0.071491	4.875756
32	0.057659	6.045454
64	0.067183	5.188453

### 2.2 Performance Data - Critical Section

Table 2: Execution Time and Speedup with Critical Section

Threads	Execution Time (s)	Speedup
1	0.635315	1.000000
2	6.664423	0.095329
4	12.303908	0.051635
6	29.000959	0.021907
8	24.990627	0.025422
10	5.889634	0.107870
12	6.047451	0.105055
16	6.623699	0.095915
20	7.340030	0.086555
32	7.197632	0.088267
64	6.885792	0.092265

## 2.3 Visualizations

### 2.3.1 Reduction Construct Performance

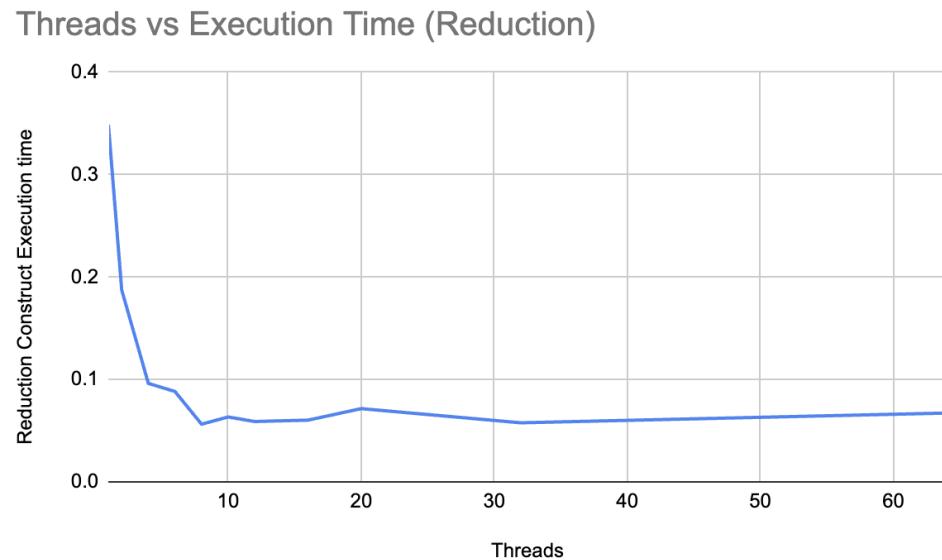


Figure 1: Threads vs Execution Time (Reduction)



Figure 2: Threads vs Speedup (Reduction)

### 2.3.2 Critical Section Performance

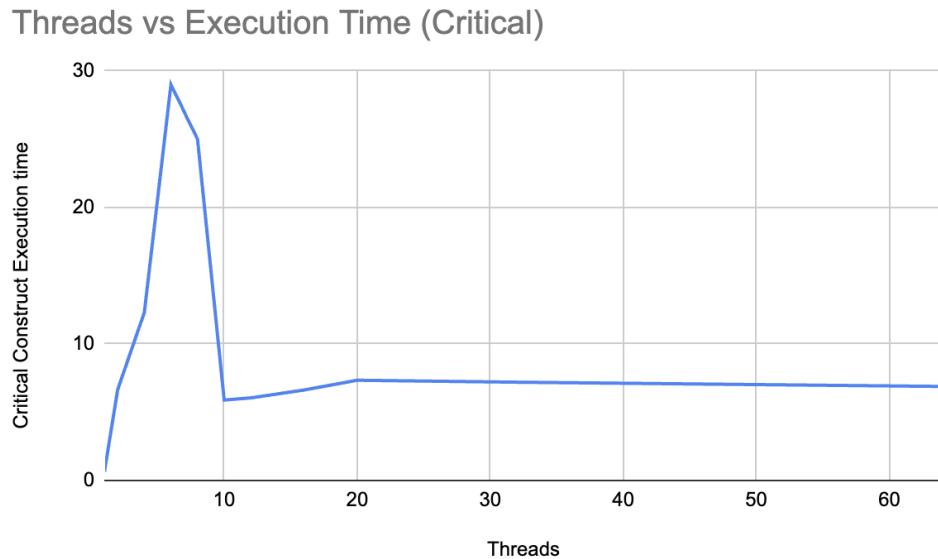


Figure 3: Threads vs Execution Time (Critical)

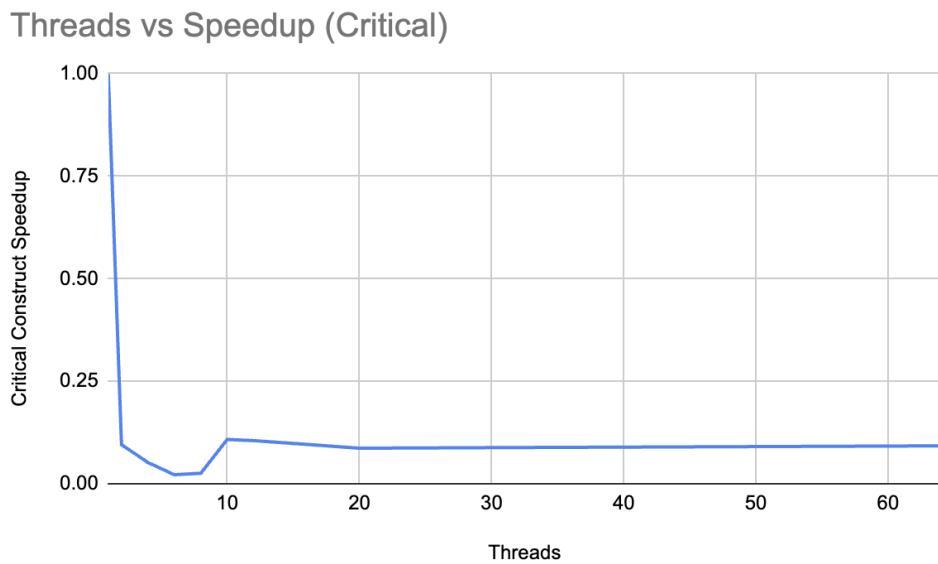


Figure 4: Threads vs Speedup (Critical)