20BCE1025
Abhishek N N
abhishek.nn2020@vitstudent.ac.in

Lab 3 - Loop sharing

1. Write a parallel program in openMP to create 8, 16 and 32 threads using runtime library routines. Construct an array of 10000 elements. Distribute the loop iterations to 32, 64, 128 concurrent threads with a chunk-size of 10, 20 and 50 using static, dynamic, guided and auto scheduling schemes. Find out the odd and even numbers global sum of 10K items. Record your execution times for the abovementioned schemes.

Code:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
	int N;
	printf("Enter N: ");
	scanf("%d", &N);
	int tid, i;
	long int odd_sum = 0, even_sum = 0;
	int A[N];
	for (i = 0; i < N; i++) A[i] = i;

	clock_t t;
	t = clock();

#pragma omp parallel for shared(A, even_sum, odd_sum) num_threads(8)
schedule(static, 10)
	for (i = 0; i < N; i++) {
		if (A[i] % 2 == 0)
		even_sum += A[i];
		else
		odd_sum += A[i];
	}
#pragma omp barrier

	t = clock() - t;
	double time_taken = ((double)t) / CLOCKS_PER_SEC;
	printf("8 threads in static: Even sum:%li Odd sum:%li\n", odd_sum,
	even_sum);
```

```
      printf("Time for static method: %fs\n\n", time_taken);
      odd_sum = 0;
      even_sum = 0;

      t = clock();

#pragma omp parallel for shared(A, even_sum, odd_sum) num_threads(16)
schedule(dynamic, 20)
      for (i = 0; i < N; i++) {
            if (A[i] % 2 == 0)
            even_sum += A[i];
            else
            odd_sum += A[i];
      }
#pragma omp barrier

      t = clock() - t;
      time_taken = ((double)t) / CLOCKS_PER_SEC;
      printf("16 threads in dynamic: Even sum:%li Odd sum:%li\n",
      odd_sum, even_sum);
      printf("Time for dynamic method: %fs\n\n", time_taken);
      odd_sum = 0;
      even_sum = 0;

      t = clock();
#pragma omp parallel for shared(A, even_sum, odd_sum) num_threads(32)
schedule(guided, 50)
      for (i = 0; i < N; i++) {
            if (A[i] % 2 == 0)
            even_sum += A[i];
            else
            odd_sum += A[i];
      }
#pragma omp barrier
      t = clock() - t;
      time_taken = ((double)t) / CLOCKS_PER_SEC;
      printf("32 threads in guided: Even sum:%li Odd sum:%li\n",
      odd_sum, even_sum);
      printf("Time for guided method: %fs\n\n", time_taken);

      return 0;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ gcc -fopenmp 1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 1
8 threads in static: Even sum:0 Odd sum:0
Time for static method: 0.002574s

16 threads in dynamic: Even sum:0 Odd sum:0
Time for dynamic method: 0.113021s

32 threads in guided: Even sum:0 Odd sum:0
Time for guided method: 0.003357s

abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 2
8 threads in static: Even sum:1 Odd sum:0
Time for static method: 0.001826s

16 threads in dynamic: Even sum:1 Odd sum:0
Time for dynamic method: 0.042402s

32 threads in guided: Even sum:1 Odd sum:0
Time for guided method: 0.003563s

abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 4
8 threads in static: Even sum:4 Odd sum:2
Time for static method: 0.002797s

16 threads in dynamic: Even sum:4 Odd sum:2
Time for dynamic method: 0.009698s

32 threads in guided: Even sum:4 Odd sum:2
Time for guided method: 0.003269s

abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 10
8 threads in static: Even sum:25 Odd sum:20
Time for static method: 0.002176s

16 threads in dynamic: Even sum:25 Odd sum:20
Time for dynamic method: 0.021007s

32 threads in guided: Even sum:25 Odd sum:20
Time for guided method: 0.003556s

abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 10000
8 threads in static: Even sum:10748832 Odd sum:11235196
Time for static method: 0.219813s

16 threads in dynamic: Even sum:8029663 Odd sum:8266358
Time for dynamic method: 0.026199s

32 threads in guided: Even sum:10090835 Odd sum:9796274
Time for guided method: 0.004288s
```

2. Write a parallel program to sort N elements in an array using OpenMP
    i. Bubble Sort

Code:

```
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

void swap();

int main(int argc, char *argv[]) {
      int SIZE = 10;
      int A[SIZE];
      for (int i = 0; i < SIZE; i++) {
            A[i] = rand() % 100;
            printf(" %d", A[i]);
      }
      printf("\n");

      int N = SIZE;
      int i = 0, j = 0;
      int first;
      double start, end;
      start = omp_get_wtime();
      for (i = 0; i < N - 1; i++) {
            first = i % 2;
#pragma omp parallel for default(none), shared(A, first, N)
            for (j = first; j < N - 1; j += 1) {
                  if (A[j] > A[j + 1]) {
                        swap(&A[j], &A[j + 1]);
                  }
            }
      }
      end = omp_get_wtime();
      for (i = 0; i < N; i++) {
      printf(" %d", A[i]);
      }

      printf("\n-------------------------\n Time Parallel= %f\n", (end -
      start));
}

void swap(int *num1, int *num2) {
      int temp = *num1;
      *num1 = *num2;
      *num2 = temp;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ gcc -fopenmp 1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
 83 86 77 15 93 35 86 92 49 21
 15 21 35 49 77 83 86 86 86 92
------------------------
 Time Parallel= 0.129913
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$
```

ii. Quick Sort

Code:

```
#include <stdio.h>

#include <omp.h>
#include <stdlib.h>

void swap(int* a, int* b) {
	int t = *a;
	*a = *b;
	*b = t;
}

int partition(int arr[], int start, int end) {
	int pivot = arr[end];
	int i = (start - 1);

	for (int j = start; j <= end - 1; j++) {
		if (arr[j] < pivot) {
			i++;
			swap(&arr[i], &arr[j]);
		}
	}
	swap(&arr[i + 1], &arr[end]);

	return (i + 1);
}

void quicksort(int arr[], int start, int end) {
	int index;

	if (start < end) {
		index = partition(arr, start, end);
```

```
#pragma omp parallel sections
          {
#pragma omp section
               {
                    quicksort(arr, start, index - 1);
               }
#pragma omp section
               {
                    quicksort(arr, index + 1, end);
               }
          }
     }
}

int main() {
int N = 10;
int arr[N];
for (int i = 0; i < N; i++) {
arr[i] = rand() % 100;
printf(" %d", arr[i]);
}
printf("\n");

quicksort(arr, 0, N - 1);

printf("Array after Sorting is: \n");

for (int i = 0; i < N; i++) {
printf(" %d", arr[i]);
}
printf("\n");
return 0;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ g++ -fopenmp 1.cpp
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
 83 86 77 15 93 35 86 92 49 21
Array after Sorting is:
 15 21 35 49 77 83 86 86 92 93
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$
```