Indian Institute of Information Technology, Vadodara

Parallel Programming(CS403)

Prof. Reshmi Mitra

Name - Dilip Puri
ID - 201351014
Collaborator - Hemant Kumar(201352026)

Lab 03

Submission Date - September 5, 2016 **Deadline** - Sep 5, 5:00 PM

---

1. Familiarize yourself with the OpenMP code given in this handout.
2. Using the command in the section on "How to measure time for a block of C-code", find the time required in the thread function call in the previous lab assignment.

```
dilip@dilip-notebook-pc:~/Desktop/sem-7/parallelprog/parallel-programming-lab/lab03$ cc -pthread time_measure.c
dilip@dilip-notebook-pc:~/Desktop/sem-7/parallelprog/parallel-programming-lab/lab03$ ./a.out
Thread 0 did 0 to 12500:  mysum=12500.000000 global sum=12500.000000
Thread 7 did 87500 to 100000:  mysum=12500.000000 global sum=25000.000000
Thread 1 did 12500 to 25000:  mysum=12500.000000 global sum=37500.000000
Thread 6 did 75000 to 87500:  mysum=12500.000000 global sum=50000.000000
Thread 3 did 37500 to 50000:  mysum=12500.000000 global sum=62500.000000
Thread 4 did 50000 to 62500:  mysum=12500.000000 global sum=75000.000000
Thread 2 did 25000 to 37500:  mysum=12500.000000 global sum=87500.000000
Thread 5 did 62500 to 75000:  mysum=12500.000000 global sum=100000.000000
Elapsed: 0.002498 seconds
Sum =  100000.000000
```

Figure 1: Elapsed time: 0.002498 seconds

3. Write a C-code using OpenMP threads to create an unbalanced load using sleep command and omp_hello.c. The sample sleep times are given below:
   (a) thread-1: 10 sec, thread-2: 5 sec, thread-3: 20 sec, thread-4: 30 sec.
   (b) Measure the total time taken for the complete execution of code with and without the additional sleep command.

```
time ./a.out
Hello World from thread = 1
Hello World from thread = 1
Hello World from thread = 0
Number of threads = 2
Hello World from thread = 1

real    0m45.005s
user    0m0.001s
sys     0m0.006s
```

Figure 2: with sleep() command execution

4. Write a threaded code using OpenMP on matrix multiplication:

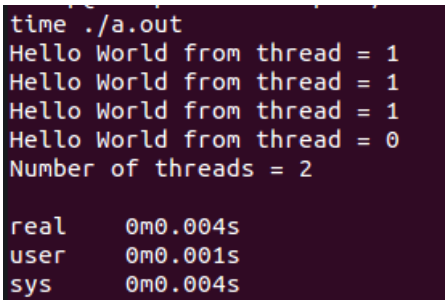

Figure 3: without sleep() command execution

(a) Take overall execution time measurement using time command for different application size and thread count for the serial and parallel code.

Table 1: Execution Time

Application Size	p=1	p=2	p=4	p=8	p=16	p=32
5x5	0.000004	0.000009	0.000050	0.000115	0.000317	0.000484
10x10	0.000017	0.000021	0.000082	0.000236	0.000263	0.000416
100x100	0.015396	0.026991	0.027809	0.021341	0.021806	0.027567
500x500	0.674168	0.787767	1.368984	1.394844	1.349535	1.441614
1000x1000	7.558727	6.348322	15.236813	14.245590	14.108841	14.337918
5000x5000						
10000x10000						

$$Speedup = \frac{ExecutionTime(p)}{ExecutionTime(serialcode)}$$

Table 2: Speedup Time

Application Size	p=2	p=4	p=8	p=16	p=32
5x5	0.000009	0.000050	0.000115	0.000317	0.000484
10x10	0.000021	0.000082	0.000236	0.000263	0.000416
100x100	0.026991	0.027809	0.021341	0.021806	0.027567
500x500	0.787767	1.368984	1.394844	1.349535	1.441614
1000x1000	6.348322	15.236813	14.245590	14.108841	14.337918
5000x5000					
10000x10000					

(b) Observe `gnome-system-monitor` output as your fire up different thread counts.

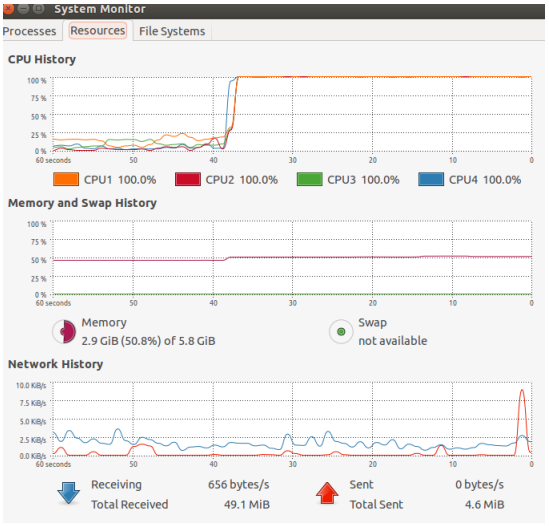

Figure 4: for 32 threads 5000x5000

(c) Use the overall execution time measurements to plot and comment upon the speed-up. Verify that:

i. Parallel execution time is more compared to the serial time when the application size is small.

ii. For large application size, initially the speedup increases as you increase the number of threads. However, for very large number of threads (e.g. 64 threads) your performance becomes much worse w.r.t. serial code and lower thread count.

5. **Multi-access threaded queue - using OpenMP** Implement a multi-access threaded queue with multiple threads inserting and multiple threads extracting from the queue. Use mutex-locks to synchronize access to this queue. Document the time for 1000 insertion and 1000 extractions each with 4 insertion threads (producers) and 4 extraction threads (consumers).(Buffer size = 250)

3 Multi-access threaded queue

1. Implement a multi-access threaded queue with multiple threads inserting and multiple threads extracting from the queue. Use mutex-locks to synchronize access to this queue. Document the time for 1000 insertion and 1000 extractions each with 4 insertion threads (producers) and 4 extraction threads (consumers).

2. Repeat above problem with condition variables (in addition to mutex locks). Document the time for the same test case as above. Comment on the difference in the times.