Navneet Nandan
ID: 201451076
4th year
B. Tech CSE

Lab 3 Report

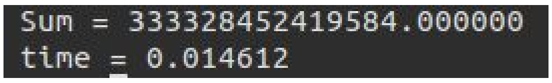
Question 2

Objective:

To measure time for a block of C-code

Observation:

Implemented the time calculation for dot product calculation implemented in last class, observed time using given methodology



```
Sum = 333328452419584.000000
time = 0.014612
```

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

int main (int argc, char *argv[])
{
int   i, n;
float array[100000], b[100000], sum;

/* Some initializations */
n = 100000;
for (i=0; i < n; i++)
  array[i] = b[i] = i * 1.0;
sum = 0.0;
clock_t start = clock();
#pragma omp parallel for reduction(+:sum) num_threads(4)
  for (i=0; i < n; i++)
    sum = sum + (array[i] * b[i]);
clock_t end = clock();

printf("   Sum = %f\n",sum);
printf("   time = %f\n",(double)(end-start)/CLOCKS_PER_SEC);
}
```

Navneet Nandan
ID: 201451076
4th year
B. Tech CSE

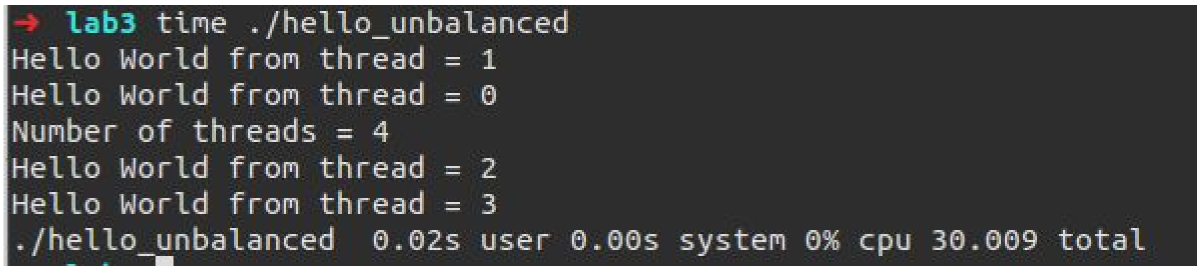
Question 3

Objective:

Use OpenMP threads to create an unbalanced load using sleep command and analyse time of execution.

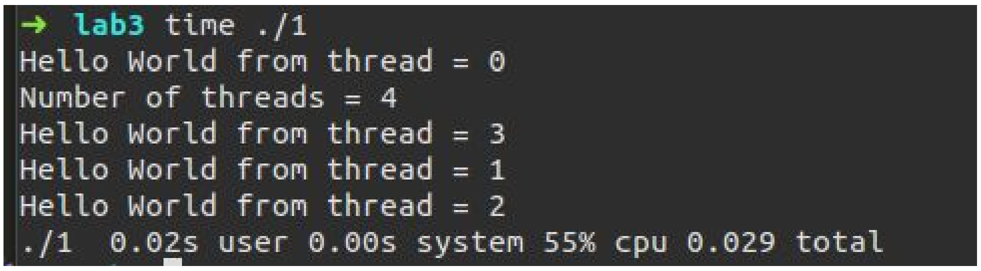
Observation:

Screenshot of execution of unbalanced load with sleep



```
➜  lab3 time ./hello_unbalanced
Hello World from thread = 1
Hello World from thread = 0
Number of threads = 4
Hello World from thread = 2
Hello World from thread = 3
./hello_unbalanced  0.02s user 0.00s system 0% cpu 30.009 total
```

Time of execution is 30.009 s.
Screenshot of execution without sleep



```
➜  lab3 time ./1
Hello World from thread = 0
Number of threads = 4
Hello World from thread = 3
Hello World from thread = 1
Hello World from thread = 2
./1  0.02s user 0.00s system 55% cpu 0.029 total
```

As there would be no sleep all threads can compete and sequence is random.

Conclusion

Execution time in unbalanced load is close to maximum time of sleep among all 4 threads, while after removing all sleep instruction all threads execution is truly parallel and execution sequence is random in nature.

Question 4

Objective:

Use OpenMP threads to do matrix multiplication by varying matrices of different size and also varying number of concurrent threads to do so.

Navneet Nandan
ID: 201451076
4th year
B. Tech CSE

Observation

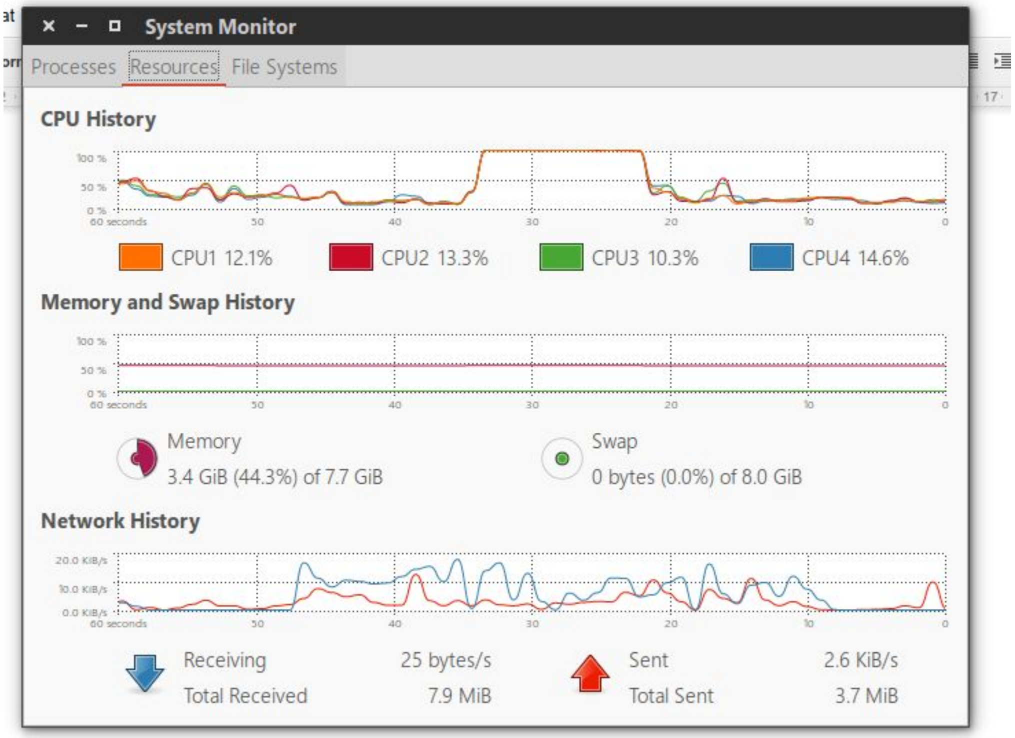
Execution Time

Size of Matrix	p=1	p=2	p=4	p=8	p=16	p=32
5x5	0.002	0.010	0.012	0.011	0.011	0.010
10x10	0.004	0.003	0.015	0.010	0.011	0.012
100x100	0.03	0.035	0.040	0.035	0.036	0.040
500x500	4.694	2.482	2.233	2.460	2.738	2.703
1000x1000	41.155	20.519	10.024	12.093	11.662	12.296
10,000x10,000				5340	5130	

Speedup

Size of Matrix	p=2	p=4	p=8	p=16	p=32
5x5	5	6	5.5	5.5	5
10x10	0.75	3.75	2.5	2.75	3
100x100	1.16	1.33	1.16	1.2	1.3
500x500	0.52	0.47	0.5	0.58	0.57
1000x1000	0.49	0.24	0.29	0.28	0.29

Navneet Nandan
ID: 201451076
4th year
B. Tech CSE



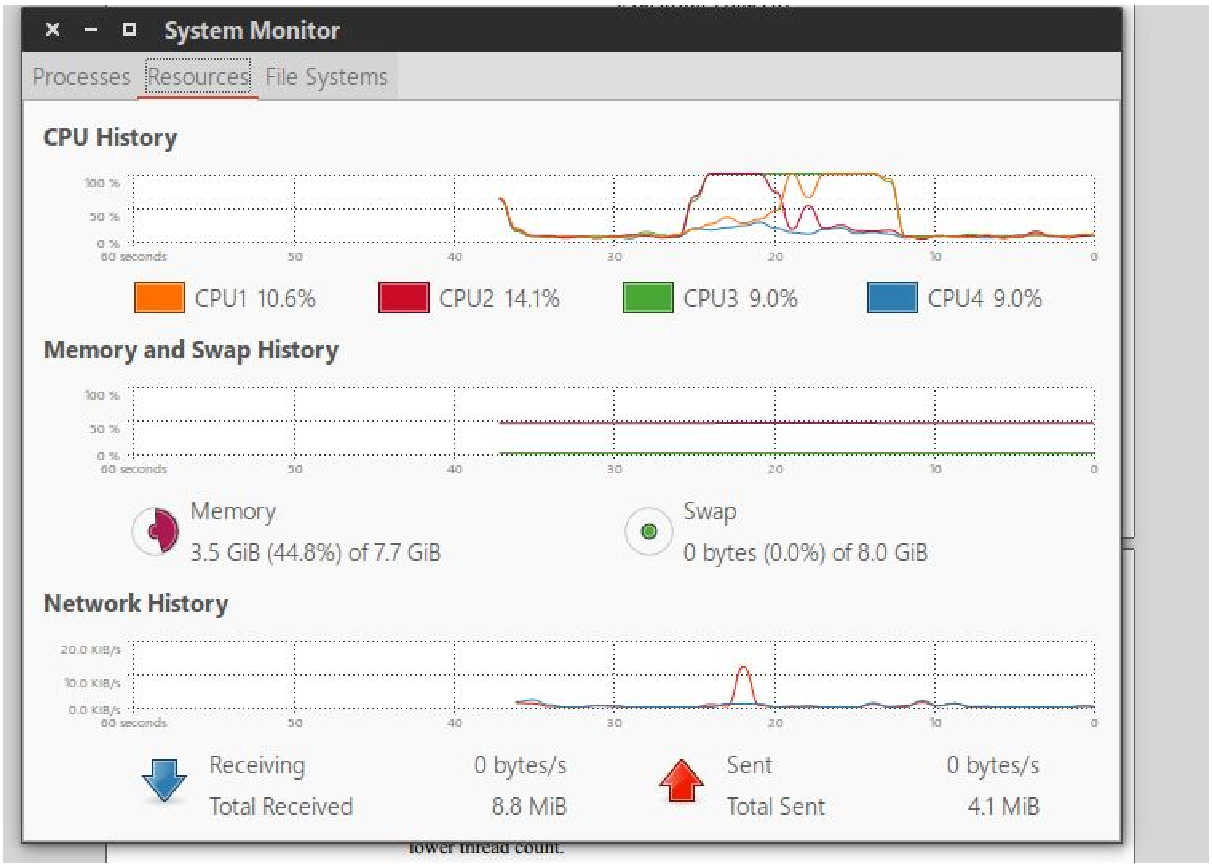
🟧 CPU1 12.1% 🟥 CPU2 13.3% 🟩 CPU3 10.3% 🟦 CPU4 14.6%

Memory
3.4 GiB (44.3%) of 7.7 GiB

Swap
0 bytes (0.0%) of 8.0 GiB

Receiving 25 bytes/s
Total Received 7.9 MiB
Sent 2.6 KiB/s
Total Sent 3.7 MiB

Gnome-System Monitor observation for matrix size of 1000 and on 4 thread



🟧 CPU1 10.6% 🟥 CPU2 14.1% 🟩 CPU3 9.0% 🟦 CPU4 9.0%

Memory
3.5 GiB (44.8%) of 7.7 GiB

Swap
0 bytes (0.0%) of 8.0 GiB

Receiving 0 bytes/s
Total Received 8.8 MiB
Sent 0 bytes/s
Total Sent 4.1 MiB

Gnome-System Monitor observation for matrix size of 1000 and on 2 thread

Navneet Nandan
ID: 201451076
4th year
B. Tech CSE

Conclusion:

Parallel execution time is more compared to the serial time when the matrix size is small. For large matrix size, initially the speedup increases as you increase the number of threads. However, for very large number of threads performance becomes much worse w.r.t. serial code and lower thread count.

Question 5

Objective:

To 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).

Conclusion:

Multithreaded Queue with 4 producer and consumer is implemented.