Lab-4 Assignment

Vaibhav Patel - 201401222
Tanmay Patel - 201401409

Hardware details:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                12
On-line CPU(s) list:   0-11
Thread(s) per core:    1
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
Stepping:              2
CPU MHz:               1264.218
BogoMIPS:              4804.38
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              15360K
NUMA node0 CPU(s):     0-5
NUMA node1 CPU(s):     6-11
```

REMARKS: Here we can see that the CPU has 6 cores in one socket. And there are 2 sockets. So we can see in the plots below that we get maximum speed up around 8 cores. Why we are not getting 12? Because It has to communicate in between the two sockets.

Question 1:

1. Context:

● Brief Description of the problem : Calculate value of pi using Monte Carlo method by generating random numbers.

● Complexity of the algorithm (serial) : If N coordinates are generated randomly then the complexity is O(N).

● Possible Speedup (theoretical) :

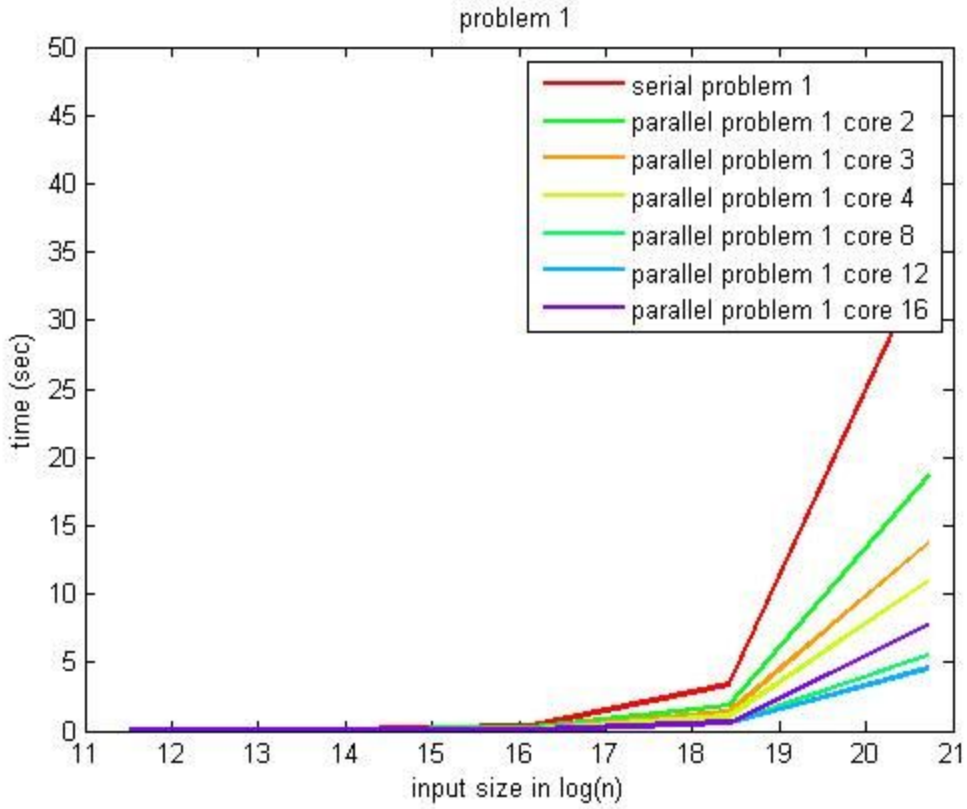
Possible speedup = 4
We are getting X.XX (maximum)
Speedup = 1 / (p / N + s) where N is no of core.
For p = 1 and s = 0 (i.e. completely parallel code) Speedup = N.
For this problem therefore speedup possible is nearly N (N = 4 in our case).

● **Optimization strategy and Problems faced in parallelization and possible solutions :** At first we used the built in c function rand() and it was worse than the serial implementation. Then we read the man page of the rand(). And then we came to know that it is not thread safe function. Then we made our own random generator and it was embarrassingly parallel code. And the speed up and efficiency was very good.(number of cores and ~1 respectively)
But our random generator's after normalizing it's mean did not converged to nearly zero. So there was an error in the value of pi. After it we read the erand48() and it is thread safe.
So now we are getting little less than theoretical speed up.

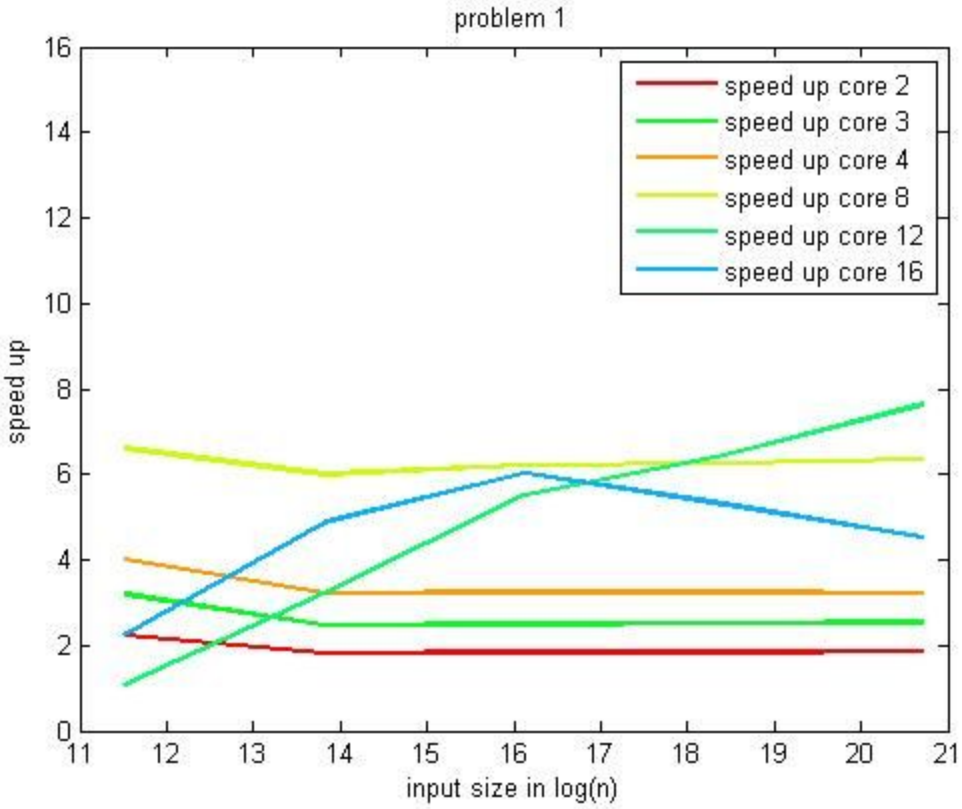
2. Input parameters and Output: Input is N where N is number of points to be generated and output is value of pi.

3. Parallel overhead time:
.02415s

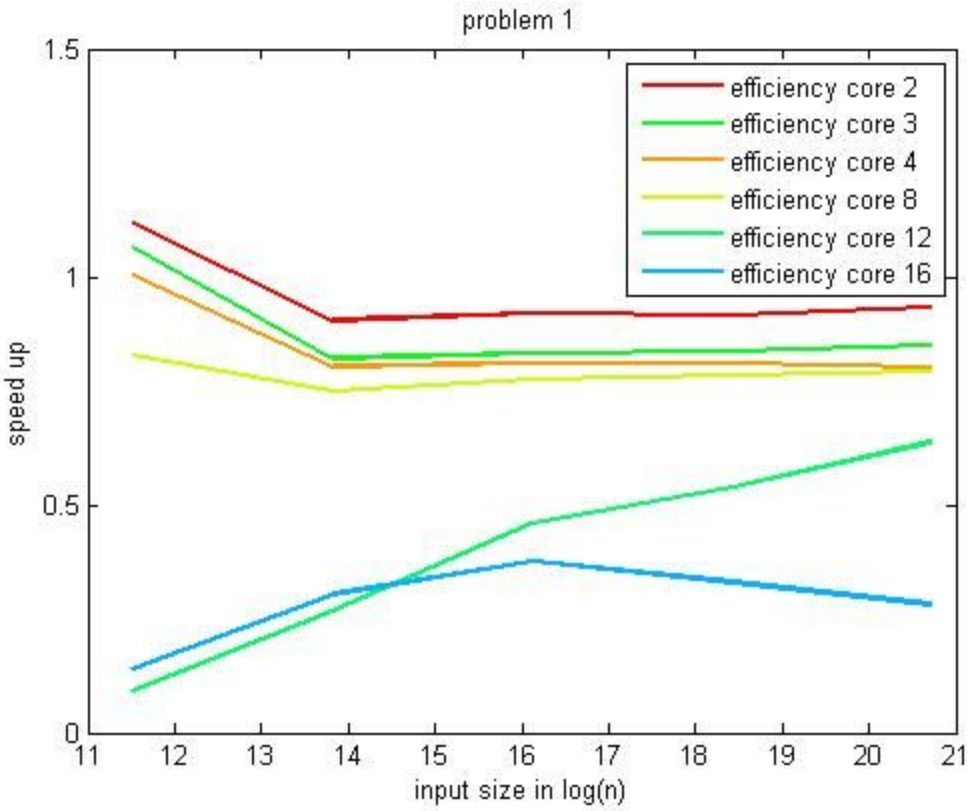
4. Problem Size vs Time:



5. Problem size vs. speedup curve.



6. Problem size vs. Efficiency.



Question 2:

1. Context:

● Brief Description of the problem : Write code for Reduction directive of pragma_omp_parallel.

● Complexity of the algorithm (serial) : Serial code is simple code to find sum of N numbers. So complexity is O(N).

● Possible Speedup (theoretical) : We can get speed up of N/log2(N) maximum.
Because there is a thing called step size. Whatever we do we can't get more than (serial time)/(step size)

● Optimization strategy : The problem is straightforward. Summation is done in serially very easily. Now we want to optimize it. But even if we use N/2 cores for the problem, we have to add those result too in the final answer.
So, We tried to add those N/2 answers using N/4 cores. And so on.

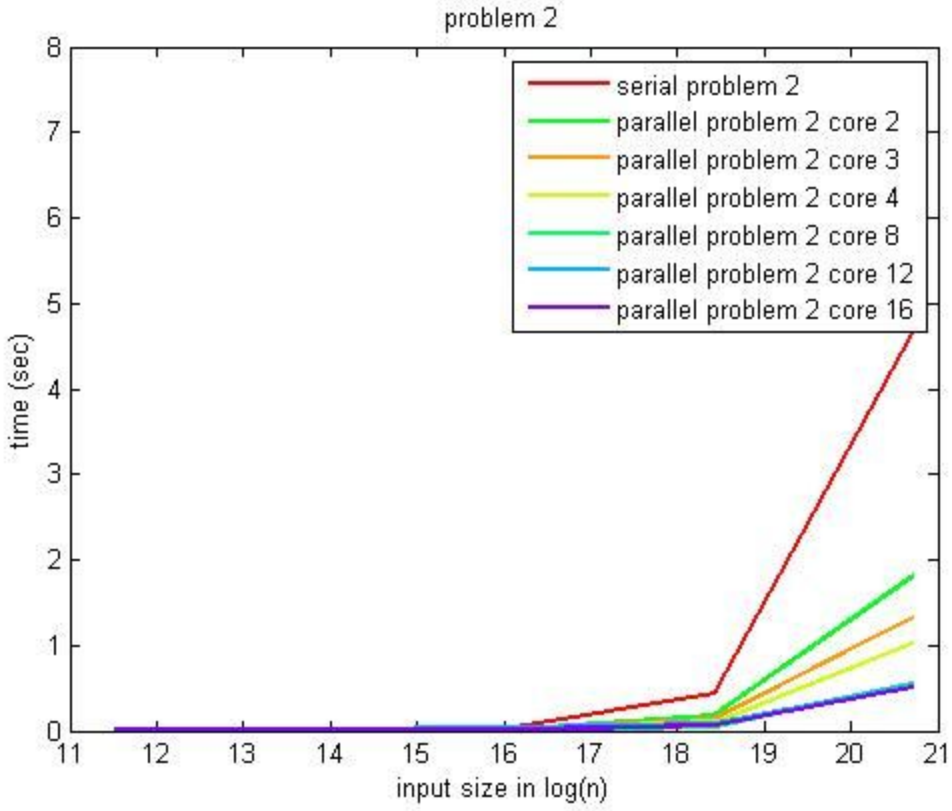
So finally we got exponential speed up.

● Problems faced in parallelization and possible solutions :
The main problem was the cache line. If we access near elements by different threads and then update it. It will update it in the RAM too. And then this will reflect in the thread which loaded this data into in its cache line. **But this problem is theoretic for us.** Because we don't have this many cores.

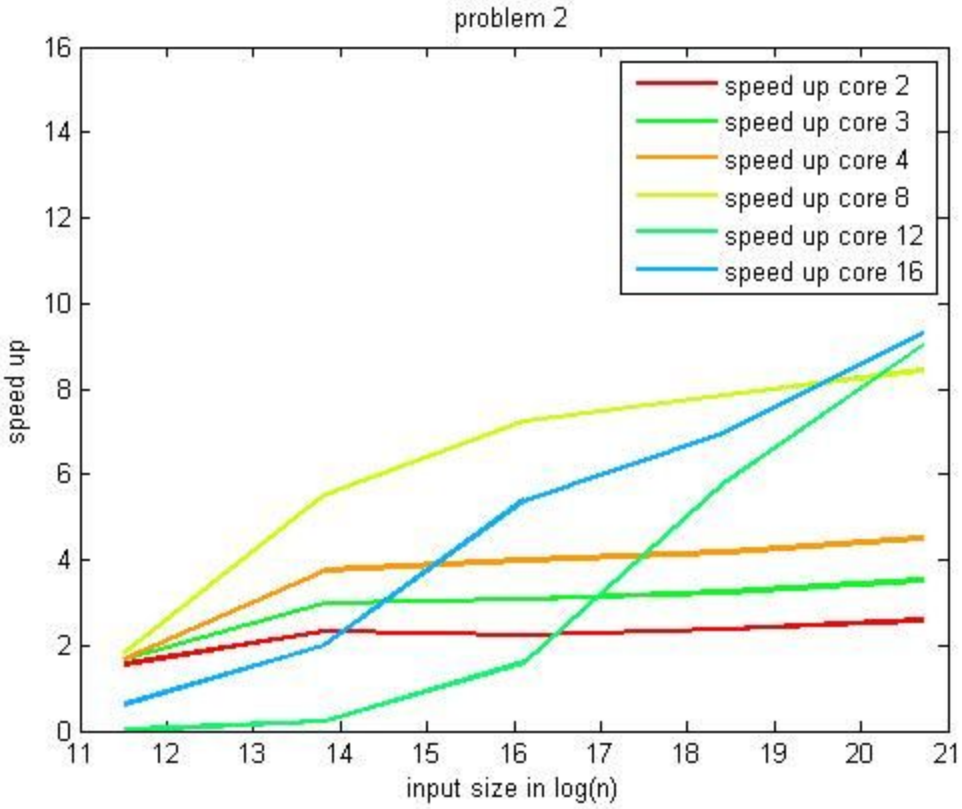
2. Input parameters and Output: Input is number of integers N and output if sum of first N positive integers.

3. Parallel overhead time:
.021483s

4. Problem Size vs Time:



5. Problem size vs. speedup curve.


problem 2
speed up core 2
speed up core 3
speed up core 4
speed up core 8
speed up core 12
speed up core 16
speed up
input size in log(n)
0
2
4
6
8
10
12
14
16
11
12
13
14
15
16
17
18
19
20
21

6. Problem size vs. Efficiency.