Q1) How many cores are here in ada?
Answer : 40 using command nproc --all

Q2) In slide no. 5 why is there no effective speedup after 38 threads? (Why is fluctuating randomly?)
Answer : In this program we have used the concept of padding the array to avoid false sharing. As there are almost 40 cores in our machine, and every core has its own cache. Two thread can run concurrently only if it is running on multiple core processor system, but if it has only one core processor then two threads can not run concurrently. So the optimal performance of the program comes when we use around 40 threads. After that the performance decreases due to false sharing of cache line.

Q3) Reason for bad performance of pi2.c?
Answer : Due to false sharing of cache line, there is also no improvement in the speed.
False sharing occurs when threads on different processors modify variables that reside on the same cache line.
In our program, the different threads require variables that are adjacent in memory (sum array) and reside on the same cache line. The cache line is loaded into the local caches of each core. Even though the threads modify different variables, the cache line is invalidated, forcing a memory update to maintain cache coherency. This reduces the performance.

Q4) What is exactly is the padding size in pi3.c and how can we decide that?
Answer : The padding size depends on the length of the cache line . We have used padding size 8 as we assume there are 64 bytes in the cache line.
The **advantage** of this method is, it is guaranteed that each successive element in the array is on different cache line. So there is no false sharing.
The **disadvantage** of using this method is that cache line size may vary from machine to machine, so I have to readjust the padding size everytime I run my code on different machine.

Q5) In slide 12 what is reduction(plus:sum)??
Answer : reduction (op : list)
It says the compiler to create a local copy of the variable in the list, that's local to each thread. Now it has to initialise local copy, so we use the identity of the operator. For example,
If op is "+" we use 0, if its "*" we use 1. So the operator defines the identity and I initialise the local copy with the identity. Every thread computes its local copy, but when every thread is done, it combines the local copy from each thread using the reduction operator into single global copy.

Q6) Find an openmp program where dynamic scheduling performs better than static.
Answer : Dynamic scheduling performs better than static scheduling when every thread has to do different work. That is when one thread is doing a lot of computation and other thread has to do very less computation i.e one thread has to do radically different work from the other thread.

So the thread with less computation finishes fast can do another computation by the time the first thread finished its computation i.e threads can dynamically do the iterations.

E.g. program : In this program each loop iteration sleeps for a number of seconds equal to the iteration number:

```
#define THREADS 4
#define N 16
int main ( ) {
  int i;

  #pragma omp parallel for schedule(static) num_threads(THREADS)
  for (i = 0; i < N; i++) {
    /* wait for i seconds */
    sleep(i);
  }
  /* all threads done */
  return 0;
}
```

In this program as we can all the threads have to radically different work so dynamic scheduling performs much better than static scheduling.

Q7) Characteristics of programs that run better with static and with dynamic and with guided.

Static :

1. Used when there is not much difference in computation performed in iterations.
2. Iterations are divided into chunks of size chunk size. The chunks are assigned to the threads statically in a round-robin manner, in the order of the thread number. The last chunk to be assigned may have a smaller number of iterations. When no chunk size is specified, the iteration space is divided into chunks that are approximately equal in size. Each thread is assigned at most one chunk.
3. Decides at compile time on how to break up loop iterations

Dynamic :

1. Used when you have radically different work from one thread to the next.
2. The iterations are assigned to threads as the threads request them. The thread executes the chunk of iterations (controlled through the chunk size parameter), then requests another chunk until there are no more chunks to work on. The last chunk may have fewer iterations than chunk_size. When no chunk size is specified, it defaults to 1.
3. Decides at run time on how to break up loop iterations

Guided :

tIn this schedule, the chuck size decreases as the program runs. It starts out relatively large and decreases to the chuck size we specify - or 1 if the chuck size is not specified. The rationale behind this scheme is that initially larger chunks are desirable because they reduce the overhead. Load balancing is often more of an issue toward the end of computation. The system then uses relatively small chunks to fill in the gaps in the schedule.

Both the dynamic and guided schedules are useful for handling poorly balanced and unpredictable workloads.