Name: Floriane Gabriel Sciper: 248112

Julien von Felten 234865

1: a) For both program, we decide to split the work equally for each thread. Using OpenMP, we have all threads running in parallel. We use also a private counter for each thread to sum (sumThread) so we don’t need to access all the time a global variable that have to be atomically accessed (meaning sequential access). The random variables are also generated in each thread. If the random generator was global, we could have some problems with the concurrency (using twice the same random number or have to wait the other thread to generate a random number). After the for loop, we add the sumThread to the general sum atomically to avoid having concurrent problem. At the end, we compute what the program needs to compute (pi or the integral computation). This means that we could parallelize most of the program, the only thing we cannot parallelize is the computation at the end, which is impossible to parallelize.

Since we only have threads in parallel and nothing in serial, we only have one phase.

b) The operations that dominate the execution time for the parallel part is in the for loop since it is where it spends the most time. Since we have to generate a new random number and we have some multiplication, this can be one of them. We don’t know how much it costs to generate the next random number in comparison with the multiplication.

c) The argument that affect the number of performance is the samples in our method. Then it is the splitSamples that divides the job among the threads. The for-loop must be done samples/num\_threads times. So, the more samples we have, the more the threads will have job to do. We would say that it is linear since each thread can work on their own and it depends only on the computation samples / num\_threads, where num\_threads are a constant. So, the big O would be O(n/C) = O(n).

d) For the speed up, we can say that we could parallelized the whole program, so the computation of the speedup using Amdahl’s Law is the number of core if we compare with a sequential run.