ASSIGNMENT 3 : PTHREADS

ANSWERS:

QUESTION: Why do you think some measurements are so erratic?

The measurements are erratic in that their values are unvarying for small values of n, but heavily improving when the “n” becomes larger. This is because, when the value of “n” is small the number of threads being used is large, and the “n” is divided into the number of threads, more than necessary naturally. When more than half of the number of tasks are multi-threaded, this leads to it taking longer than the optimal time to actually merge the different threads, and thereby dipping towards then end, when there aren’t enough input values to use to compute.

Secondly the when the intensity is low for iteration, the values remain almost constantly low because the time per function is very less, and thereby no matter how many threads are run, they are all serially added to the global function, and there is no difference in the speedup as the function itself is not one to take too long to run. Hence there is no scope for parallelism, causing not much speedup. These are the parts of the erratic function observed.

QUESTION: Why is the speedup of low intensity runs with iteration-level synchronization the way it is?

In the low intensity iteration level runs, the key pattern observed is that the values are never changing and low. The reason for this is the time of function running when intensity is reduced. When the time of function running is low, the amount of time for parallelism is low, which makes the parallel part of the program low too. The proportion of the other part is equal or higher, causing a speedup which is almost not significant.

The above is a consequence of Amdahl’s law.

1/((1-P)+(P/N)

Here P would become lower, when the program has low intensity, and hence would cause 1-P to be higher and P/N to be higher. Hence the program would not achieve as much parallelism as it would when run with large intensity.

QUESTION: Compare the speedup of iteration-level synchronization to thread-level synchronization. Why is it that way?

The speed in thread for low intensity is also visible, since in threads the joining of the many processes for larger n values after computing fractions of them individually, is unlike in iterations, where the process directly adds to the total sum.

Hence in this case, for the larger numbers, the parallel part of the program tends to be higher, although it will add a small overhead of merging time. This factor causes a higher than usual speedup. Iteration level speedup is less significant for this reason. They do remain the same for smaller values of n as expected, since the number of threads exceed the value, and hence don’t require parallelism. In brief, the cause for this is the amount of time the code spends in summing and executing. The intensity code will have a longer time to wait when a lot of threads queue up after short function execution times, to

take the mutex to add their values to it . Instead, the method of threads, causes them to run in parallel without much queuing.

Again, according to Amdahl's law, the parallel part of the program is not significant in comparison to the non-parallel part in the iteration in THIS CASE (as it queues up often and causes higher wait time – non-parallel part), unlike thread, which doesn't .

Question: Compare performance at 16 threads across the different synchronization modes. Why are the speedup this way?

The speedup at 16 threads is almost non significant up until the point of 10,000 intensity. This is because the intensity. This again is a result of the running time. When the intensity is as low as even 100, the 16 threads will not matter in changing the intensity. Although, after this having 16 threads doesn't make a difference or rather even reduces the speedup without the right granularity, since it causes the time of merging to be of nuisance when seen from the perspective of speedup.

Throughout this process, we see the importance of granularity in both thread and chunk. If the granularity is not correctly selected, the speedup will not have a linear improvement. As noticed, the speedup in general is highest when the granularity is a perfect division of "n" into the number of threads. In many cases there is a similarity between the thread and the chunk performance for a given n and a given intensity.

chunk (n=10000 intensity=100), in this case, with granularity, 100 although we notice a similar speedup in both cases, we notice that the there is a sharp increase in the case of chunk, when the number of threads goes higher, this is because with a higher intensity and good granularity, "chunks" best features can be computed.

In this case

Iteration on the other hand would be more benefitted in cases with high "n" value, low intensity value and lower number of threads,and high granularity. This would mean it would run most of it in a single loop with less threading and lower function run time, and all the additions are done in 1, without an issue.

chunk (n=1000000 intensity=1000)

At the above value and at 1 granularity a better speedup in thread than in chunk for 1 granularity .This is because when the granularity is less, there is a sharp change in the number of times the global summation variable is accessed. In this case the thread is better , since it doesn't queue up to access the global sum value, making it perform better.

Question: For thread level synchronization, compare the performance at 16 threads of different n and intensity. Why are the plots this way?

At 16 threads, for different values of n, we can see that the n values vary significantly. The following are our conclusions. When the number of "n" is large enough, the number of threads at 6 will cause a

larger speedup than for a lower value of n and intensity, which has less scope for parallelism . A classic example.

thread (n=100 intensity=1000) vs thread (n=10000 intensity=1000)

Is a classic example. When n is small and intensity are equal against a high n value, we notice that the speedup is significantly lower in threads, in spite of 16 threads being deployed. The key reason for this is that the processes aren’t divided well enough, and n is too small causing the merge task between the 16 threads to be long. Finally, we notice that this entire process varies with intensity, in that when the intensity is higher , the parallelism is more beneficial, since each process runs for longer and by amdahl’s law the function runs longer and hence the parallel part is longer , causing the speedup to be higher. This is the key observed phenomenon.