YANAMANDRA SHARATH CHANDRA (585953)

Results of performance measurements:

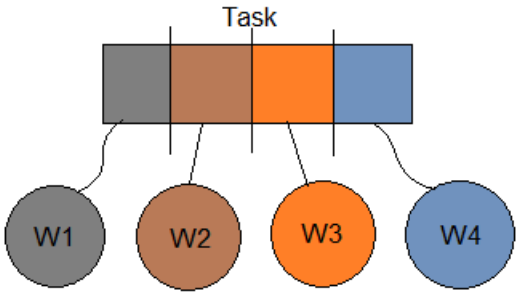
Well, In the SPO some part of the code must be executed sequentially, so no matter how many threads we use, we have to pay that time. In the code it is updating the global optimum.

Remaining part can be parallelized, so every time we double the number is threads, we should have maximum half the time needed to execute this part. So, given if parallel part with many threads takes less time that of sequential, the program does not scale well no more. If we add time to setup the threads, when we use many threads the time could also be increased.

In the Performance measurements the graphs scale well up to some number of cores and later they measure with small changes or increases, instead of decreasing. This may have some couple of reasons like overheads. We will discuss them accordingly where it may occur in the code,

Source of overheads:

1. Inter process communication: For instance, if we have 4 workers and a task is divided in between the workers. When we use parallel programming, there is a high demand for exchange of data/ results/ control information. When this exchange happens, there is a requirement of communication. so finally, more communication and less work done, then this overhead occurs. In the code this could occur at dividing into fork and merge them at join.

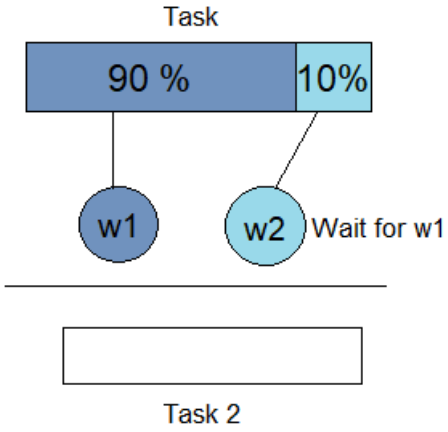


2. Idling: When non uniform distribution of data is occurred, idling occurs.

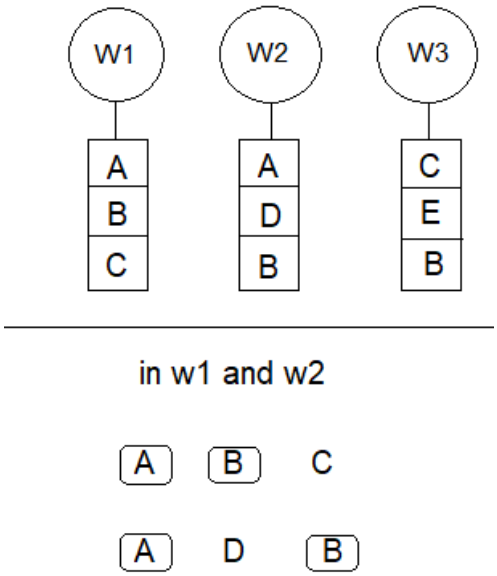
   Case 1: If a task is divided between two workers (W1, W2) in the ratio 90:10. W1 will take more time to compute than W2 which leads to waiting of W2 until other worker finishes. Because the next task must be completed together.

   Case 2: In some cases, the output of one worker will be input of the other worker. This will also tend to wait of one of the workers.

In our case in multithread code I used mutex locks, that is guard lock and a barrier for synchronization. This could be one of the occurrences of the overhead.



3. Excess computation: Occurs when redundancy of computation occurs. When the two workers compute same instruction without knowing, excess computation occurs. (load imbalance, synchronization)



Now about the resultant graphs, the parallel part takes less time with respect to sequential fraction so increasing the number of threads does not improve much and sometimes the time may also increase. Hence, performance is collapsed.

Another point is taken into consideration is about number of particles enough for specific number of threads. In my case it is efficient till 64 cores for 10000 particles. Later the threads overlap and the performance is collapsed and hence the time of computation is increased and the graph is also not stable.

In the program I used ‘struct’ variable for each particle, so overhead is accessing the memory of particle position while updating the position of particle.