Mezemir Gebre

Parallel Programing Skills

A screenshot of a cell phone

Description automatically generated

The above screenshots show the output for trap-notworking an trap-working program which computes the definite integral from 0 to pi of sin x and return the value. The difference between the working version and the unworking version of this program is the line that contains the pragma for parallelizing the loop that is computing the integral. In the unworking version, we used “#pragma omp parallel for private(i) shared (a, n, h, integral)”, it lacks a reduction clause for the accumulator variable integral so the calculate result is not correct. However, for the working version, we added a reduction clause on line 38, as a separate line for the accumulator variable (integral) as shown here “37 #pragma omp parallel for \ 38 private(i) shared (a, n, h) reduction(+: integral)” so we got the correct result 2 as you can see.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

The above screenshots are for the barrier program. The first output is without the barrier pattern so as we can see from the output, the pattern is going as “before, after, before, after” and so on this shows that the execution continued before all of the threads complete the parallel section of code. Whereas the next screenshot above is an output I got with the barrier pattern uncommented. As you can see, when we use the barrier pattern, all threads complete a parallel section of code or (before the barrier) before executing the after the barrier section or the section after any parallel section code.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

The above two screenshots are the output for the masterWorker program. The first one is without the pragma on line 24 so we have only the output from the master #0 thread because commenting the pragma doesn’t let the workers or the rest of the threads fork and execute different block of code. However, in the next screenshot as you can see, the master thread (#0) forks and executed one block of code then the workers forked and executed different block of code.