Coding 2: Numerical integration

October 21, 2024

1 Numerical Integration (60 pts)

Take your sequential code to compute numerical integration from the previous assignment. Modify it to do the numerical integration in parallel. Output the value of the integration to stdout and the time taken to compute it to stderr.

1.1 Implementation with openMP (30 pts)

Implement with OpenMP loop construct. Make sure to activate OpenMP in GCC by passing `-fopenmp` to the compiler and linker. Note that if you omit this parameter, the code will probably still compile. But its execution will be sequential. You can control the number of threads in OpenMP in two ways: the environment variable `OMP_NUM_THREADS` or by calling the `omp_set_num_threads` function. The number of threads to use will be passed to the program as a parameter to main. So use the `omp_set_num` threads function call to set the number of threads. Loop scheduling in OpenMP can be controlled either by specifying schedule(runtime) in the parallel for construct and specifying a `OMP_SCHEDULE` environment variable, or by using the `omp_set_schedule` function. The scheduling policy is passed to the program as a parameter to main, so use the `omp_set_schedule` function call.
Submission: a **main_omp.cpp** code with the abovementioned implementation.

1.2 Implementation using pthreading (30 pts)

Implement the multithreaded version using pthreads. You should divide the workload across multiple threads. Each thread will compute a portion of the integration, and the results will be accumulated at the end.
Hint: here are the steps to achieve this:

1. Thread Data: Create a structure to store data for each thread.
2. Thread Function: Implement a function that each thread will execute to compute its portion of sum.
3. Thread Creation: Divide the iterations among threads and compute the partial sums in parallel.
4. Thread Joining: Collect the partial sums from all threads and combine them into the final result.

Submission: a **main_pth.cpp** code with the abovementioned implementation.

2 Benchmarking on centaurus (40 pts)

Evaluate your implementations on Centaurus under the following scenarios and report the time of each scenario in a png file.

- **Question:** Evaluate the openMP implementation using `f1` and different values of `n` (from 10^1 to 10^8) and `intensity` (from 1 to 10^4). Show the change of time across `n` and `intensity` with a fixed 4 threads.

- **Question:** Evaluate the pthreading implementation using `f1` and different values of `n` (from 10^1 to 10^8) and `intensity` (from 1 to 10^4). Show the change of time across `n` and `intensity` with a fixed 4 threads.
- **Question:** Evaluate the openMP implementation with different number of threads (1, 2, 4, 8, 16, 32, 64, 128) using `f1` and `n` of 10^7) and `intensity` of 10^4). Show the change of time across the changing number of threads.
- **Question:** Evaluate the pthreading implementation with different number of threads (1, 2, 4, 8, 16, 32, 64, 128) using `f1` and `n` of 10^7) and `intensity` of 10^4). Show the change of time across the changing number of threads.

Submission: **time_{1, 2, 3, 4}.png** for questions 1 - 4.