P. Hadjidoukas

Set 2 - Numerical Integration and Multithreading

Issued: March 15, 2023

Question 1: Parallel Numerical Integration

The value of an integral $\int_a^b f(x)dx$ can be approximated by computing its Riemann sum:

$$S = \sum_{i=1}^{n} f(x_i^*)\Delta x$$

where $\Delta x = x_i - x_{i-1} = (b-a)/n$, x_i^* some point in the interval $[x_{i-1}, x_i]$, $x_0 = a$ and $x_n = b$. The *midpoint approximation* uses, in the Riemann sum, the middle point $\bar{x}_i = \frac{(x_{i-1}+x_i)}{2}$ of each interval $[x_{i-1}, x_i]$. In this question you will implement and parallelize this method.

a) The `integral_seq.c` file calculates sequentially the integral $\int_1^4 f(x)dx$, where $f(x) = \sqrt{x} \cdot ln(x)$, using Riemann sum with the midpoint approximation.

Parallelize the code by starting several threads to compute the Riemann sum. Make sure you do not introduce race conditions and verify your implementation by comparing the final result with that computed by the serial program. Note that each thread should handle a different interval of the integral.

The parallel code can be found in the file `integral_mt.c`.
The number of threads can be specified as a runtime argument to the executable, e.g. the command: `./integral_mt 4`, runs the parallel code with 4 threads. If not specified, the code uses 2 threads by default.

b) Choose an appropriate number (n) of intervals and check how the wall-clock time for computing the integral decreases with respect to the number of threads. Plot the time versus the number of threads and report your observations.

The measurements were conducted on a single compute node with 24 cores. Each experiment was repeated multiple times and the lowest time was taken. In Figure 1, we observe that the execution time decreases as the number of threads increases. However, the code fails to scale on more than 20 threads, a behavior which is better depicted in the speedup (strong scaling) plot of Figure 2. The observed performance degradation is attributed to overheads introduced by the thread scheduler of the operating system. This issue can be resolved by applying thread pinning, i.e. binding each thread to a specific processor core. This technique is automatically supported by OpenMP and will be discussed in the related lectures and exercises.

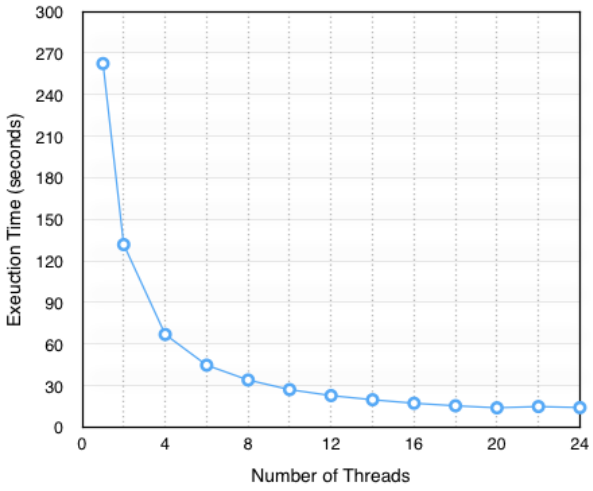

Figure 1: Execution time vs Number of threads.

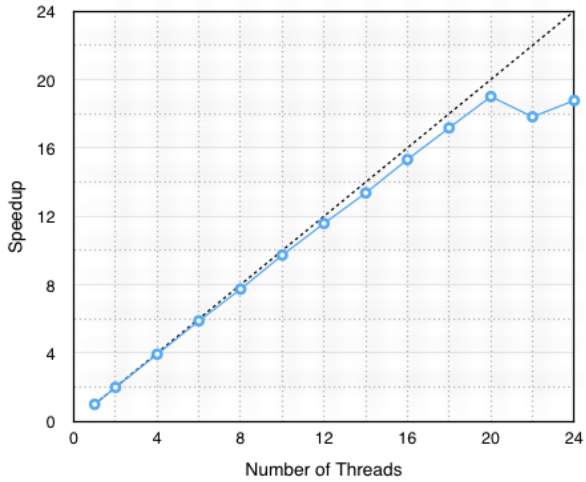

Figure 2: Strong scaling plot.