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:

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

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.

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 time measurements can be performed on any computer system of your preference (e.g. personal laptop). In this case, you should report the hardware/software configuration of that system (i.e. number and type of cores, operating system and compiler).

Question 2: Bug hunting and performance optimization

In the following code, `do_work()` is an external thread-safe function with a considerable execution time (e.g. several seconds) and returns a floating-point number that differs each time. Each thread calls the function and add the returned value to the local variable counter.

```c
#include <stdio.h>
#include <pthread.h>

extern float do_work();  // implemented elsewhere
float counter = 0;

void *func (void *arg)
{
  for (int i = 0; i < 10; i++)
    counter += do_work();

  return NULL;
}

int main (int argc, char * argv[])
{
  pthread_t id[4];
  for (int i = 0; i < 4; i++)
    pthread_create(&id[i], NULL, func, NULL);

  printf("counter=%f\n", counter);
  return 0;
}
```

1. Report any possible issues of the code.
2. Modify the code to resolve those issues and explain your solution.

Constants and functions:

```c
PTHREAD_MUTEX_INITIALIZER
pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
pthread_mutex_lock (pthread_mutex_t *mutex);
pthread_mutex_unlock (pthread_mutex_t *mutex);
pthread_join(pthread_t thread, void **retval);
pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*func)(void *),
    void *arg);
```