AMATH 483 / 583 (Roche) - Homework Set 6

Due Monday May 19, 5pm PT

May 12, 2025

Homework 6 (75 points)

1. (+20) **Length of a graph** of $f(x)$ on $[a,b]$ is $L = \int_a^b \sqrt{1+(f'(x))^2}dx$. Given $f(x) = ln(x) - \frac{1}{8}x^2$ on $[1,6]$. (a) Find the length of f on the interval analytically. You will need this to calculate the error in your codes. (b) Write a C++ code parallelized using the **C++ threads** library that numerically evaluates the length of this function on the interval using Riemann sum. Your code should accept the number of points used to partition the interval and the number of threads to spawn. (c) Plot the strong scaling efficiency on 1,2,4,8,16 threads of your code for $n = 1.e8$ partition points, i.e. time versus thread count. (d) Plot the *log* of the numerical error for $10, 100, ...1.e6$ partition points (increasing by factors of $10x$ each time). Submit your code and plots.

2. (+15) **Length of a graph** of $f(x)$ on $[a,b]$ is $L = \int_a^b \sqrt{1+(f'(x))^2}dx$. Given $f(x) = ln(x) - \frac{1}{8}x^2$ on $[1,6]$. (a) Write a C++ code parallelized using **MPI** that numerically evaluates the length of this function on the interval using Riemann sum. Your code should accept the number of points used to partition the interval. (b) Plot the strong scaling efficiency of your code on $1, 2, 4, 8, 16, 32$ processes for $n = 1.e8$ partition points using Hyak. (c) Plot the *log* of the numerical error for $10, 100, ...1.e6$ partition points (increasing by factors of $10x$ each time). Submit your code and plots.

3. (+30) **Elevator scheduler**. I provide `hw6-elevator.cpp` and snippet `hw6-elevator.hpp` codes. Edit the header file to implement the two functions:

   ```
   void elevator(int id); void person(int id);
   ```

   Please don't change the global variables, but feel free to add what is needed for your functions. You will submit the file `hw6-elevator.hpp`. The exercise is to write a threaded elevator scheduler for a 50-story business building with 6 elevators. Up to 7500 people may visit the building in a day. Your functions should use the C++ threads library to simulate the behavior of the elevators and visitors of the building. The driver code takes as input the number of people to service (hint: makes it a bounded buffer). After servicing all person requests, have each elevator report the number of passengers it transported, and be certain that your code exits cleanly. Please print events to *cout* in a manner that is thread safe. For each person, report their ID and current floor and destination floor (i.e. *Person 0 wants to go from floor 7 to floor 9*), report the elevator number they get on (i.e. *Person 1 entered elevator 0*), report when they arrive at their target floor (i.e. *Person 98 arrived at floor 18*). For each elevator, report the direction of movement from current floor to next floor (i.e. *Elevator 2 moving from floor 12 to floor 19*), join all the elevator threads once all visitors have arrived at their floors, and report success (i.e. *Job completed!*).

   - randomly select the current floor and destination floor for each person
   - I let each elevator start from the ground floor if it matters
   - issue a person thread in the *detach* state (i.e. *thread(person, i).detach();*)
   - each elevator will be it's own joinable thread
   - keep track of the current floor and direction of each elevator
   - keep track of the destinations of the visitors who are waiting for elevators on each floor
   - determine which elevator should respond to a given request, based on its current location, direction, and the number of passengers it is already carrying
   - the maximum occupancy of the elevators is fixed, say 10
   - move the elevators up or down to their next destination, picking up and dropping off passengers as necessary

- print events to screen as they occur
- add some sleep functions to mimic the action of the elevators moving between floors

4. (+20) **Communication Bandwidth**. (a) Use the following template declaration to write a **MPI** C++ function using point-to-point communication that implements broadcast. Submit the function in file `my_broadcast.hpp`. (b) Measure and plot the performance of your function and the `int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)` function on messages of size $8B, 16B, ..., 256MB$ bytes (doubling the message size each instance). Do this exercise using Hyak on 4 and 32 MPI processes, and make a plot of your measurements with bandwidth (bytes per second) on the y-axis, and the message size in bytes on the x-axis. Submit your plot.

   - ```
     template <typename T>
     void my_broadcast(T* data, int count, int root, MPI_Comm comm);
     ```
     ```