Homework 1

Emma Fountain

I. Implementation

I implemented the 1d-wave equation as explained by section 1 of the homework-1 prompt using the C programming language.

There are 2 different versions of my code, one which simply calculates the appropriate values for the wave, and one which saves the y-values of the wave at each time step so that we can later render a wave animation. To save the y-values to disk, simply define the `SAVE` macro when compiling or remove it to disable saving.

II. Timing Study

I timed the code using the linux `time` command and I tested this code both on my local machine and on MSU's HPCC, with specs as shown in Table I. In order to (attempt to) take advantage of MSU HPCC's greater compute resources I used the cluster with the latest processor (amd22) and scaled up memory and number of cores by 4 compared to my laptop. I ran each version of the code 5 times, using the "user" time output from the `time` command then averaged the runtimes to get final estimates as shown in Table II below.

My laptop actually significantly outperformed the MSU HPCC cluster. This is likely because my code is written completely in serial and there is no RAM bottlenecking so the additional HPCC resources are not actually being utilized. Due to this, since my processor's clock speed of 4.6 GHz is significantly faster than the HPCC's 2.6 GHz processor, it makes a lot of sense that my laptop outperformed the HPCC.

III. Wave Animation

Rather than create an animation using C, I opted to save the generated y-values of the wave to disk then use python with matplotlib to generate a GIF. I saved the y-values at each iteration as entries in a csv file called `y.csv` which I then loaded into python using the `numpy` package. Once this data was loaded, I use `matplotlib` to generate a GIF animation. Due to the large number of time-steps, I only sampled every 1000 steps, but this proved more than adequate to produce a high-quality animation (included as wave.gif in the tarball).

IV. Improvements

The most obvious improvement would be parallelizing computations at each x-position when updating position, velocity, and acceleration.

TABLE I
System Specs

	Laptop	**MSU HPCC**
OS	Arch Linux	Ubuntu
CPU model	Intel i7-11800H	AMD EPYC 7763
CPU cores	16	64
CPU speed	4.6 GHz	2.6 GHz
RAM	16 GiB	64 GiB

TABLE II
Timing Study Results (seconds)

	Laptop	**HPCC**
Saving	60.6172	102.3548
No Saving	9.5982	15.3694

V. Reproducing Results

A. Timing Study (Unix-Based)

If you are using a unix-based OS, reproducing results should be fairly easy.

1) Ensure gcc is installed
2) Uncompress and navigate to the project folder
3) Run `chmod u+x experiment.sh` to allow execution of the `experiment.sh` file.
4) Run `./experiment.sh`

B. Timing Study (Windows)

If you are using windows the only difference is that the `experiment.sh` file will not work so you must run each experiment yourself.

1) Ensure gcc is installed
2) Uncompress and navigate to the project folder
3) Run `gcc wave.c -o wave.out -lm -D'SAVE'` to compile the program to save its output
4) In powershell run `Measure-Command { ./wave.out}` 5 times
5) Run `gcc wave.c -o wave.out -lm` to compile the program without saving output
6) In powershell run `Measure-Command { ./wave.out}` 5 times

C. Animation

If you would like to use the `y.csv` file to generate the animation the following steps should be OS-agnostic.

1) Ensure `numpy` and `matplotlib` are installed in your current environment
2) Run `python makeVis.py`

The new animation should be found in a file named `wave.gif`