CS 581: High Performance Computing
Emory University
Spring 2025
Prof. Tianshi Xu

Homework # 3

Mitchell Scott
(mtscot4)

1 Environment

List the environment you used to run the code (Operating System, CPU, RAM, etc.).

Solution. The local environment where I was running the code was on a 2022 MacBook Pro with Apple M2 chip, 8 GB of RAM running macOS Sequoia 15.3.2.

2 Runtime Comparison

Compare CPU time with the GPU time.

Solution. To be consistent, I didn't comment out the writing to the file. Although this would speed it up, I didn't want to mess up the code. As a result, the CPU time took 4132 ms. The GPU time took 3594 ms.

3 Implementation

Discuss your Implimentation.

Solution. For the CPU implimentation, we essentially did 3 for loops nested - first the frame we were generating, then the particle we were looking at, then the dimension of that particle. Within the 3 nested for loops, we also had to see if the dimension was x, y, or z, and if we were looking at the $z-$component, we had to factor in gravity, and after the update, we had to make sure that if the $z-$ component was negative, we changed the location and added a inelastic bouncing coefficient. This means for the CPU code, at the deepest level, we have 3 for loops and 2 if statements all nested.
However, for the GPU code, we don't have to have as many for loops! This is because we have different threads operating on different particles, so we can eliminate the for loops for particles and dimension. This is accomplished because we wrapped all of the threads into two kernel functions - one that just updated the $x-$ and $y-$coordinates (as we don't have to factor in gravity) and the other that updates the $z-$ component, which factors in gravity, and also checks if the particle went below the ground, which needs to be accounted for. So because of the kernel functions, we have only one for loop for the frame.

4 Bottlenecks

What is the performance bottleneck of your parallel implementation?

Solution. One of the biggest performance bottlenecks of CUDA programming is the fact that we first have to cudaMalloc and then cudaMemcpy all of that data onto the device. Once everything is computed on the GPU, that data is then cudaMemcpy back into the programming. While the GPU might run faster than the CPU code, moving the large amount of data is very time consuming.
Additionally, since GPU resources might be limited for students, we decided to run on Google COLAB. Since we are using T4 GPU on Colab for free, there might be some slowing down on Google's side, as well as the transition of data from my computer to the cloud.

Acknowledgements

I would like to acknowledge that I worked in the office on Thursday, March 18 and March 25.