HPCSE II - Exercise 5

Anian Ruoss

May 12, 2019

Task 2

The results from running `nbody_naive.cu` (with increased output precision for verification) are displayed in Listing 1.

Listing 1: Output from running `nbody_naive.cu`.

```
Net Force: 0.000000139824
Absolute Force: 66480053.035935938358
Time: 16.09551728s
```

We apply the following optimizations:

- We introduce the temporary variables `x_force`, `y_force`, and `z_force` to reduce the number of writes to global memory from $\mathcal{O}(N)$ to $\mathcal{O}(1)$ per thread.

- We introduce the temporary variables `x_mIdx`, `y_mIdx`, `z_mIdx`, and `m_mIdx` to reduce the number of reads from global memory from $\mathcal{O}(N)$ to $\mathcal{O}(1)$ per thread.

- We create the arrays `x`, `y`, `z`, and `m` of size `BLOCKSIZE` in shared memory. This amounts to $\texttt{BLOCKSIZE} \cdot 4 \cdot 8$ bytes ≈ 32 kilobytes of shared memory which is still below the total amount of shared memory per block of 49152 bytes (obtained by running `deviceQuery.cpp` from `https://github.com/NVIDIA/cuda-samples/blob/master/Samples/deviceQuery/deviceQuery.cpp`). As a consequence, we reduce the number of reads from global memory from $\mathcal{O}(\texttt{BLOCKSIZE} \cdot N)$ to $\mathcal{O}(N)$ per block.

- We replace $^1/_{\texttt{sqrt}(\ldots)}$ with a call to $\texttt{rsqrt}(\ldots)$ which puts less pressure on the ALUs.

- We move the multiplication with `m_mIdx` out of the for-loops due to distributivity of multiplication and addition.

- We add 10^{-16} to the sum of squared distances to avoid dividing by zero for the case $i = m$ thereby also avoiding divergence.

- We introduce the temporary variable `tmp` to avoid computing the same term three times.

- We unroll the inner loop.

All optimizations were fairly straightforward to implement and we present the output of our optimized solver in Listing 2.

Listing 2: Output from running `nbody_opt.cu`.

```
Net Force: 0.000000144482
Absolute Force: 66480053.035935945809
Time: 2.23435090s
```

Comparing listings 1 and 2 we observe that the net and absolute forces are identical (up to an acceptable tolerance) with a speedup of ≈ 7.2.

A note on correctness: since the task asks for the optimization of the given N-Body solver without changing the problem size we do not explicitly handle the case where the problem size N is not evenly divisible by the number of threads per block[1]. However, our code can easily be extended for this purpose by enclosing the accesses to global memory on lines $38 - 41$ with `if (b + threadIdx.x < N)` and by adding `if (N <= b + i) continue;` to line 47.

[1] We also note that `nbody_naive.cu` does not handle this case either.