1 Task 1: Heat2D on CUDA

To be submitted separately as optional bonus task before May 27, 2019.

2 Task 2: Optimizing the N-body problem

2.1 Introduction

After studying the supplied naive reference solution code and while doing some initial research, I stumbled on the article [1]. I then decided that it would be rewarding to compare a complete rewrite of the code using warp shuffles with an implementation using shared memory instead of doing the ablation study somewhat prescriptively suggested by the benchmark table published on Piazza.
Thus, two implentations (warp-based, shared memory-based) of an improved force calculation kernel where written and compared in terms of obtained performance. The more efficient one then was submitted as the active one with my solution.

2.2 Improvements common to both implementations

The following straightforward improvements were applied to both kernel implementations:

- using thread-local registers for intermediate results (see below) and summing up forces during interactions
- using the built-in math function `rsqrt` instead of `1/sqrt`
- saving the intermediate scalar forces for reuse after calculating the cubed body-body distance
- avoiding divergent code by allowing $F_i = \sum_{\substack{1 \leq j \leq N \\ i \neq j}} \frac{m_i m_j \mathbf{r_{ij}}}{\|\mathbf{r_{ij}}\|^3} \simeq \sum_{1 \leq j \leq N} \frac{m_i m_j \mathbf{r_{ij}}}{(\|\mathbf{r_{ij}}\|^2 + \epsilon^2)^{\frac{3}{2}}}, \quad 0 < \epsilon \ll 1$

2.3 Warp-based implementation

The beauty of a warp-based implementation is that it doesn't require any shared memory declarations and operations. Instead, the threads within a warp (in hardware so far always of size 32, but described by the built-in variable `warpSize`) exchange register information directly. This makes for a simple implementation with an inner loop over the warp which can be unrolled. The downside of using warp exchanges at the register level is that no vector data types (e.g. `double3`) can be used an thus any potential benefit those might bring has to be forfaited.

2.4 Shared memory-based implementation

This implementation was heavily inspired by NVIDIA's own best practice implementation [2]. The key idea is to process the body-body interactions in square tiles (figure 1) where the required data is loaded into shared memory in a synchronized manner before starting calculation on a tile. For simplicity, the square tile side length was chosen to be equal to `blockSize.x`. Another performace-improving choice made is to use the types `double3` and `double4` internally in the kernel so that memory accesses would coalesce. Any `#pragma unroll` $n, n \in \{2, 4, 8, 16, 32\}$ statements in the inner body-body interaction loop did only have detrimental effects.

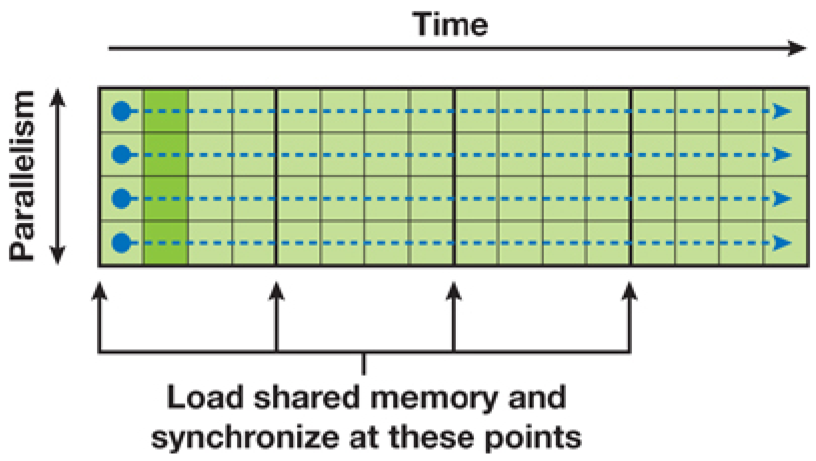

Figure 1: Tiling and shared memory logic. Image credit: NVIDIA Corporation [2].

2.5 Results

2.5.1 Warp-based implementation

Experiments were run for `blockSize.x` $\in \{256, 512, 1024\}$ with `blockSize.x` $= 512$ offering the fastest runtime as shown in listing 1.

Listing 1: Task 2: Warp-based force calculation kernel output.

```
Net Force: 0.000000
Absolute Force: 66480053.035555
Time: 2.87512449s

Batch Job Summary Report for Job "nbody_opt" (13789234) on daint
--------------------------------------------------------------------------------------------------------------
Submit               Eligible              Start                 End                  Elapsed     Timelimit
-------------------  -------------------   -------------------   -------------------  ----------  ----------
2019-05-10T22:29:41  2019-05-10T22:29:41   2019-05-10T22:39:58   2019-05-10T22:40:20    00:00:22    00:02:00
--------------------------------------------------------------------------------------------------------------
Username    Account     Partition   NNodes   Energy
----------  ----------  ----------  ------   --------------
class04     class01     normal           1           joules
---------------------------------------------------------------
gpusecs  maxgpusecs        maxmem            summem
-------  ----------  ------------  ----------------
3              3      326107136         631242752
---------------------------------------------------------------
```

2.5.2 Shared memory-based implementation

Experiments were run for `blockSize.x` $\in \{256, 512, 1024\}$ with `blockSize.x` $= 1024$ offering the fastest runtime as shown in listing 2.

Listing 2: Task 2: Shared memory-based force calculation kernel output.

```
Net Force: -0.000000
Absolute Force: 66480053.035555
Time: 2.30858409s

Batch Job Summary Report for Job "nbody_opt" (13809329) on daint
--------------------------------------------------------------------------------------------------------------
Submit               Eligible              Start                 End                  Elapsed     Timelimit
-------------------  -------------------   -------------------   -------------------  ----------  ----------
2019-05-11T16:15:40  2019-05-11T16:15:40   2019-05-11T16:16:16   2019-05-11T16:17:29    00:01:13    00:02:00
--------------------------------------------------------------------------------------------------------------
Username    Account     Partition   NNodes   Energy
----------  ----------  ----------  ------   --------------
class04     class01     normal           1           joules
---------------------------------------------------------------
gpusecs  maxgpusecs        maxmem            summem
-------  ----------  ------------  ----------------
2              2      326107136         631242752
---------------------------------------------------------------
```

2.6 Discussion

Both implementations show a slight aberration from the published naive reference implementation's total force value. However, after checking with the responsible lecturer on Piazza this was deemed acceptable.
While both kernel implementations reach the 'passing grade with bonus' treshold, only the shared memory-based kernel reaches the distinction treshold with a runtime of `2.31s` and thus is submitted as the master result..

References

[1] Harris, Mark: *CUDA Pro Tip: Do the Kepler Shuffle*, NVIDIA Developer Blog, Feb 3, 2014, `https://devblogs.nvidia.com/cuda-pro-tip-kepler-shuffle/`, last visited on May 11, 2019.

[2] Nyland, Lars; Harris, Mark; Prins, Jan: *Chapter 31. Fast N-Body Simulation with CUDA*, NVIDIA GPU Gems 3, Aug 12, 2007, `https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch31.html`, last visited on May 11, 2019.