**MAP55616. GPU Programming with CUDA**

**Assignment 2**

Luning Zhang ([zhangl14@tcd.ie](mailto:zhangl14@tcd.ie))

24th, Apr, 2023

# Task 1 – CPU calculation

I have used the default n, m, and p to test the performance of the CPU computing version of the code, and the results obtained are as follows (Only shows 10 rows of average temperature here):

**zhangl14@cuda01**:**~/A2**$ ./cpu -a -t

RunTime of CPU calculation is 0.000268 s

Row 0: 0.007685

Row 1: 0.015324

Row 2: 0.022964

Row 3: 0.030604

Row 4: 0.038244

Row 5: 0.045883

Row 6: 0.053523

Row 7: 0.061163

Row 8: 0.068803

Row 9: 0.076442

Row 10: 0.084082

... ...

... ...

... ...

# Task 2 – CPU and GPU calculation (single precision)

Also use the default matrix for testing, we can get the time like:

**zhangl14@cuda01**:**~/A2**$ ./float -a -t

CPU time: 0.000266 s

GPU time:

Allocation time: 0.000188 s

Compute time: 0.00019165 s

Calculate averages time: 0.000017 s

Transfer time: 0.000018 s

Here is the comparison of the Average temperature.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CPU Calculation** | **GPU**  **Calculation** |  | **CPU**  **Calculation** | **GPU**  **Calculation** |
| Row 0 | 0.007685 | 0.007956 | Row 16 | 0.129921 | 0.134538 |
| Row 1 | 0.015324 | 0.015868 | Row 17 | 0.137560 | 0.142450 |
| Row 2 | 0.022964 | 0.023779 | Row 18 | 0.145200 | 0.150361 |
| Row 3 | 0.030604 | 0.031690 | Row 19 | 0.152840 | 0.158273 |
| Row 4 | 0.038244 | 0.039602 | Row 20 | 0.160480 | 0.166184 |
| Row 5 | 0.045883 | 0.047513 | Row 21 | 0.168119 | 0.174095 |
| Row 6 | 0.053523 | 0.055425 | Row 22 | 0.175759 | 0.182007 |
| Row 7 | 0.061163 | 0.063336 | Row 23 | 0.183399 | 0.189918 |
| Row 8 | 0.068803 | 0.071247 | Row 24 | 0.191038 | 0.197829 |
| Row 9 | 0.076442 | 0.079159 | Row 25 | 0.198678 | 0.205741 |
| Row 10 | 0.084082 | 0.087070 | Row 26 | 0.206318 | 0.213652 |
| Row 11 | 0.091722 | 0.094982 | Row 27 | 0.213958 | 0.221564 |
| Row 12 | 0.099362 | 0.102893 | Row 28 | 0.221597 | 0.229475 |
| Row 13 | 0.107001 | 0.110804 | Row 29 | 0.229237 | 0.237386 |
| Row 14 | 0.114641 | 0.118716 | Row 30 | 0.236877 | 0.245298 |
| Row 15 | 0.122281 | 0.126627 | Row 31 | 0.244517 | 0.253209 |

# Task 3 – Performance improvement (single precision)

The GPU version involves CUDA memory allocation, initialization of the matrices in device memory, running the heat propagation in parallel using CUDA kernels, and then optionally calculating the average temperature for each row. Finally, the average temperature is copied back to host memory from device memory. It also records and displays the timings for each major step.

CUDA Kernels: There are several CUDA kernels, each performing a different operation:

init\_matrices\_kernel: Initializes the matrix on the GPU.

propagate\_heat\_kernel: Calculates the next matrix based on the current one.

swap\_matrices\_kernel: Swaps the current and next matrices.

calculate\_average\_kernel: Calculates the average temperature per row.

Test the code for different sizes, use n=15360,m=15360 and p=1000, here is the result:

**zhangl14@cuda01**:**~/A2**$ ./float -t -n 15360 -m 15360 -p 1000 -c

GPU time:

Allocation time: 0.001531 s

Compute time: 77.51101685 s

Calculate averages time: 0.008272 s

Transfer time: 0.000043 s

For the 2d grid in the GPU version, try using different numbers of threads per block and in each of the x and y directions, the running time comparison is as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **BLOCKSIZE\_x** | **BLOCKSIZE\_y** | **Allocation** | **Compute time** | **Cal\_averages** | **Transfer** |
| 8 | 8 | 0.001528 s | 87.35 s | 0.015452 s | 0.000041 s |
| 8 | 16 | 0.001515 s | 57.48 s | 0.015477 s | 0.000052 s |
| 16 | 8 | 0.001512 s | 94.04 s | 0.013991 s | 0.000050 s |
| 16 | 16 | 0.001550 s | 75.46 s | 0.014025 s | 0.000051 s |
| 32 | 16 | 0.001531 s | 77.51 s | 0.008272 s | 0.000043 s |
| 16 | 32 | 0.001522 s | 65.31 s | 0.013914 s | 0.000041 s |
| 32 | 32 | 0.001582 s | 75.56 s | 0.008286 s | 0.000050 s |

In the above cases, it seems that the BLOCKSIZE with the best performance is 8x16.

# Task 4 – Double precision testing

Using the best block size 8x16 of the single-precision version, comparing the single-precision calculation with the double-precision, the results are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Allocation** | **Compute time** | **Cal\_averages** | **Transfer** |
| float | 0.001515 s | 57.48 s | 0.015477 s | 0.000052 s |
| double | 0.002945 s | 64.97 s | 0.020028 s | 0.000068 s |

After making these changes, compare the runtime, speedup, and accuracy of the double-precision version to the original floating-point precision version. Note that the speedup is not significant because GPU double-precision calculations are slower than single-precision calculations. Also, accuracy may not improve significantly if calculations do not involve very large or very small numbers that would benefit from the increased precision of double.