CS239 Programming Exercise 2

Junelle Rey C. Bacong
Department of Computer Science
University of the Philippines Diliman
bacong.junelle@gmail.com

29 September 2019

1 The Exercise

Consider three rectangular matrices $\mathbf{A} \in \mathbb{R}^{n \times k}$, $\mathbf{B} \in \mathbb{R}^{k \times m}$ and $\mathbf{C} \in \mathbb{R}^{n \times m}$. We want to analyze the parallel execution of a simple matrix multiplication i.e. $c_{ij} = \sum_i^n \sum_j^m a_{ij} * b_{ji}$ in two different ways:

- `matmul_rec_glob` where the matrices are stored in the global memory
- `matmul_rec_shared` where tiling algorithm and shared memory are employed

2 Preliminaries

I used my personal computer with `GeForce GTX 1080 Ti` graphics card and 3584 total cores. Its maximum block size (in 2D) is $(1024, 1024)$ while its maximum grid size is $(2.1 \times 10^9, 1024)$. The total global memory of my device is at most 11GB. In my experiments, I varied the total amount of memory M_{GB} needed to store these three matrices and then compared different kernel speedup. Given n, k, m, the total memory size M_{GB} needed for the computation can be determined from

$$M_{GB} \propto n \times k + m \times k + n \times m \tag{1}$$

Also, the kernel speedup T_s is measured relatively as

$$T_s = \frac{t_{global}}{t_{shared}} \tag{2}$$

where t_{global} and t_{shared} are the execution times of `matmul_rec_glob` and `matmul_rec_shared`, respectively. In the experiments, I used a constant block dimension of 32×32 and a variable grid dimension of $ceil(n/32) \times ceil(m/32)$ because I wanted to exploit the maximum available threads per block of my hardware that is pegged at 1024. I also chose the tile widths 32, 48, 64 and 70 arbitatrily while keeping in mind the maximum shared memory per block of my device that is 48kB.

3 Results and Discussion

In Figure 1, we see that the speedup T_s is greater than 1, which means from Equation 2 that $t_{global} > t_{shared}$. This result can be attributed to the fact that global memory access is much slower than that of shared memory access [Kirk and Hwu, 2017]. The application of tiling algorithm with shared memory access made the execution time of the kernel `matmul_rec_shared` much faster, giving a speedup of at most 10.

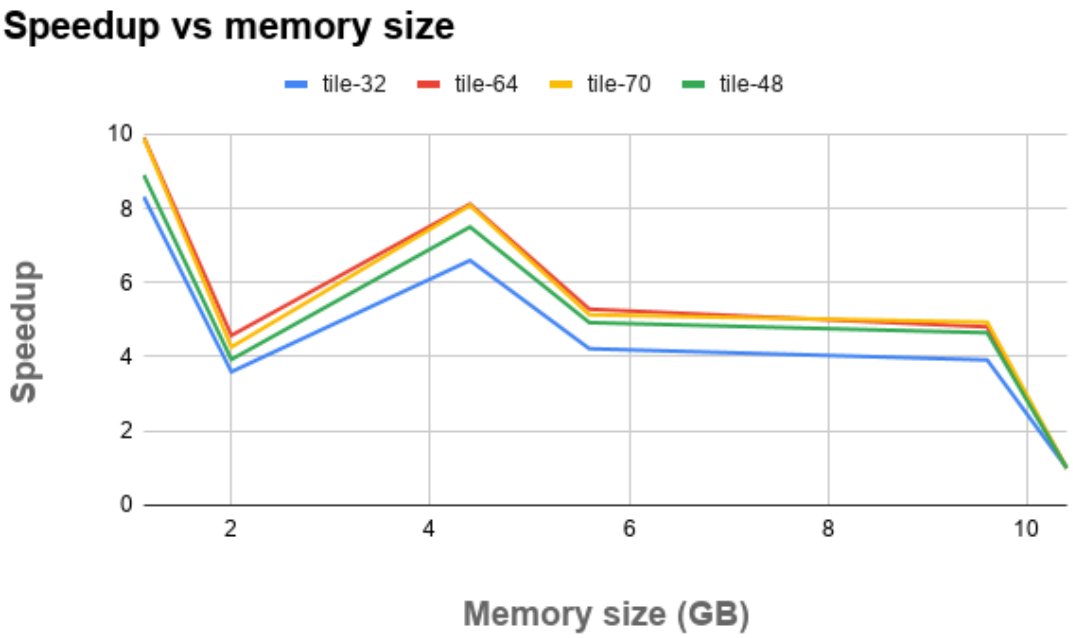

Figure 1: Comparison in performance speedup for different tile width.

For comparison also, two different tile widths were initially used to assess the performance of the `matmul_rec_shared` kernel. In Figure 1, 64-tile width kernel obtained a much faster execution time compared to its 32-tile width variant. The dimension of the tile determines the factor by which the global memory accesses is reduced. From a CGMA of 1 for `matmul_rec_global`, we expect a much higher CGMA from using `matmul_rec_shared` kernel that is proportional to the tile width used [Kirk and Hwu, 2017]. In our case, using 32 and 64 tile increases the kernels' CGMA to 32 and 64, respectively. Consequently, this significant increase in the CGMA is only directly proportional to the speedup in the execution time and does not equate to it.

Trying out other tile widths greater than 32 showed that increasing its value does not significantly increase the speedup of the tiled kernel as shown in Figure 1, especially for tile width equals to 70. This could be from the fact that tile widths should be a multiple of 32 in order to exploit the parallelism per warp of threads inside the hardware. In the case of tile width 32 and 64 for instance, a significant difference in the speed up can be observed, with the latter performing better than the former. Tile widths 48 and 70, however, do not have a significant difference from the performance speedup of tile widths 32 and 64, respectively.

Although they have bigger shared memory for storing sub-elements of matrices **A** and **B**, their parallel thread execution is limited from the warp size of the hardware.

We next obtained the throughput of different kernels. In this context, we define throughput simply as the number of data transferred and processed during the execution of the kernel. Figure 2 shows the throughput for different kernel programs. We observe that `matmul_rec_glob` has yielded that lowest throughput among the three kernels. This is because each thread of this kernel is reading the same global memory each iteration at a much slower rate. Unlike in the `matmul_rec_shared`, a subset of the global memory is being read once, and then transferred to a much accessible shared memory inside the block thread.

Between the kernel variants of `matmul_rec_shared` with tile width 32 and 64, the latter obtained a much higher throughput. This is again because of a much higher CGMA score that is proportional to its tile width. Interestingly, the throughput of both variants of `matmul_rec_shared` decreases as the memory size M_{GB} increases. This implies that the execution time of the kernel program becomes significantly slower as more data are being processed.

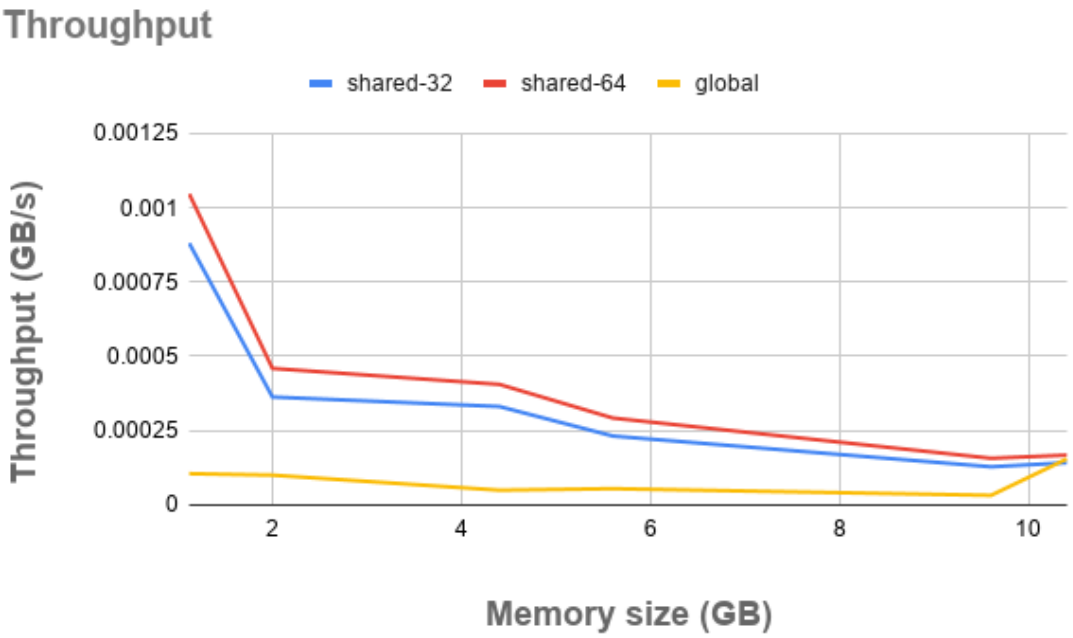

Figure 2: Throughput of different kernels.

4 Conclusion

We showed how matrix multiplication can be parallelized in two ways. Clearly, `matmul_rec_shared` is much faster than `matmul_rec_global` because of faster shared memory access inside the thread block. Employing a tiling algorithm with the use of the shared memory resulted to a much faster execution time. Aside from the amount of shared memory that can be used for

computation, the performance of the kernel is also affected by the warp size of the hardware. We have also observed a significant difference between tile widths 32 and 64, but not with other tile dimension. The same performance has been observed among the kernels for their experimental bandwidth, with `matmul_rec_shared` with tile width 64 having the highest throughput.

References

[Kirk and Hwu, 2017] Kirk, D. and Hwu, W.-m. (2017). Programming massively parallel processors: a hands-on approach. Morgan Kaufmann, an imprint of Elsevier.