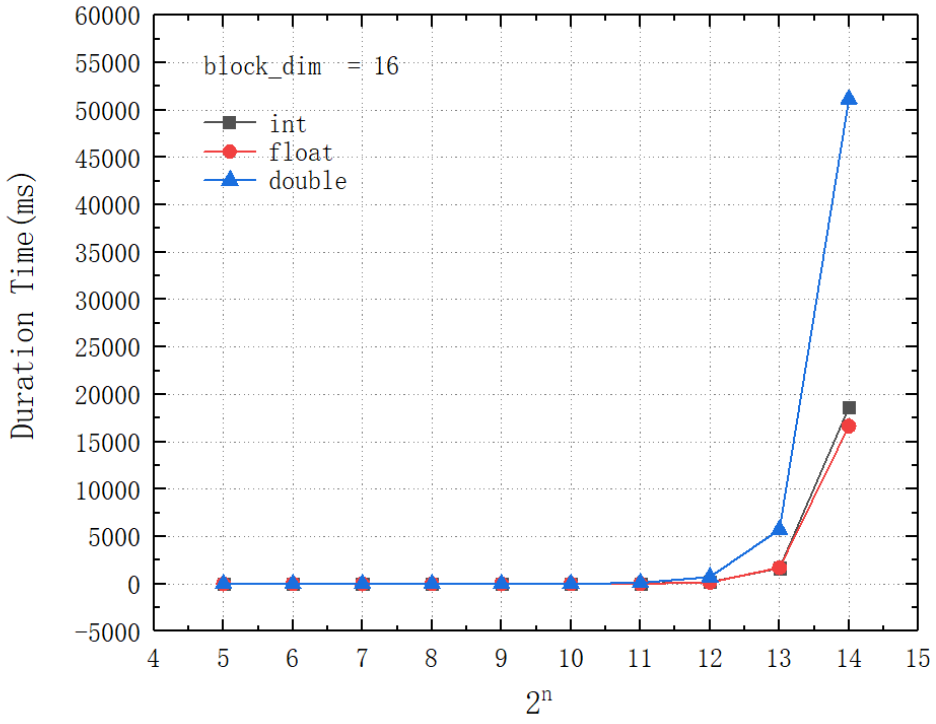
HW05 Task1 b



HW05 Task1 c

2^13			
block_dim	int	float	double
16	1795.6	1685.5	5736.3
24	1718.9	1090.6	5249.1
32	1101.8	1022.0	5526.6

By testing with block_dim values of 16, 24, and 32, we observed that increasing the block_dim generally improves performance for float and int data types. Surprisingly, for double precision data types, a block_dim of 24 provides good performance.

HW05 Task1 d

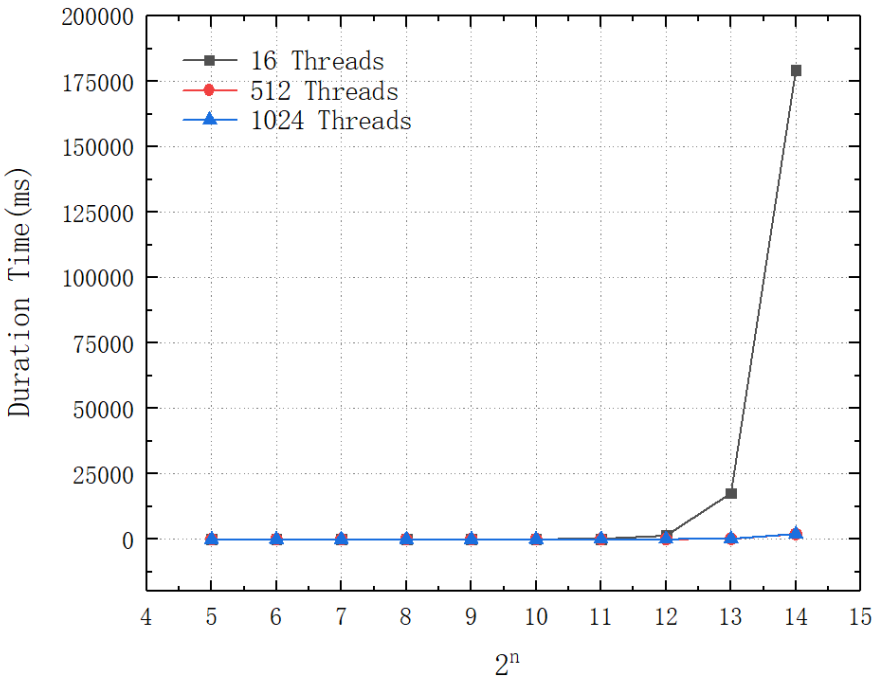
1. **float type:** Typically, matrix multiplication using the float data type achieves the best performance on the GPU as GPUs are better suited for single-precision floating-point operations.
2. **double type:** Matrix multiplication using the double data type is generally faster on the

GPU compared to the CPU, but slower than matrix multiplication with the float data type. GPUs have lower computational capabilities for double-precision floating-point numbers, resulting in lower performance.

3. **int type:** Matrix multiplication using integer data types generally performs less efficiently than floating-point types, as GPUs are better suited for floating-point operations. Consequently, the performance is typically lower.

HW05 Task1 e

HW04 Task1



HW05 task1 performs better than HW04 task1. Here are the explanations:

1. **Data locality:** Tiled matrix multiplication takes advantage of shared memory, which reduces global memory access. This results in fewer memory stalls, leading to better performance.
2. **Parallelism:** Tiled multiplication can be efficiently parallelized using a 2D grid of thread blocks, enabling more concurrent execution compared to naive multiplication.
3. **Optimized memory access patterns:** Tiled multiplication optimizes memory access patterns by loading smaller tiles into shared memory. This reduces the number of cache

misses, improving overall performance.

HW05 Task1 f

The HW02 task1 takes more than 10 minutes. Here are the explanations:

1. **Hardware Acceleration:** GPUs are optimized for parallel processing, making them more efficient for tasks like matrix multiplication, where many operations can be parallelized.
2. **Parallelism:** The GPU code may utilize parallelism to a greater extent, enabling faster computation for large matrices.
3. **Memory Access:** GPUs typically have high memory bandwidth and optimized memory access patterns, which can be advantageous for matrix operations.
4. **Algorithmic Improvements:** HW05 includes algorithmic and code optimizations that result in improved performance by utilizing shared memory.

HW05 Task2