Exercise 4

To compile: unzip our uploaded code, and run `make` inside `code/`. The slurm scripts are stored inside `code/slurm/`. To debug: run the debug outputs (`*.dbg`) and attach gdb to respective pids

4.1 Reading

4.2 Matrix multiply — parallel version using MPI

4.3 Matrix multiply — scaling process count

The program of subsection 4.2 was tested on its scalability with an increasing number of processes. The times and speedups in Table 1 were observed for 2000x2000 matrices with 100 repetitions.

	GFLOPS64/s		t/ns		Speedup
	mean	std	mean	std	
nodes					
1	1.425	0.212	11.532	2.050	1.000
2	2.138	0.207	7.555	0.756	1.526
4	2.316	0.030	6.910	0.091	1.669
6	3.399	0.050	4.709	0.071	2.449
8	4.450	0.066	3.596	0.055	3.207
10	5.251	0.161	3.050	0.094	3.781
12	5.899	0.550	2.740	0.298	4.209
14	6.521	0.772	2.496	0.360	4.621
16	6.963	1.038	2.361	0.431	4.885

Table 1: Scaling by Processes in numbers

For this Example a step is visible before 6 pocesses, most likeley because here only one node is used and we are reaching its maximum. Starting with 6 processes, this maximum is increased.

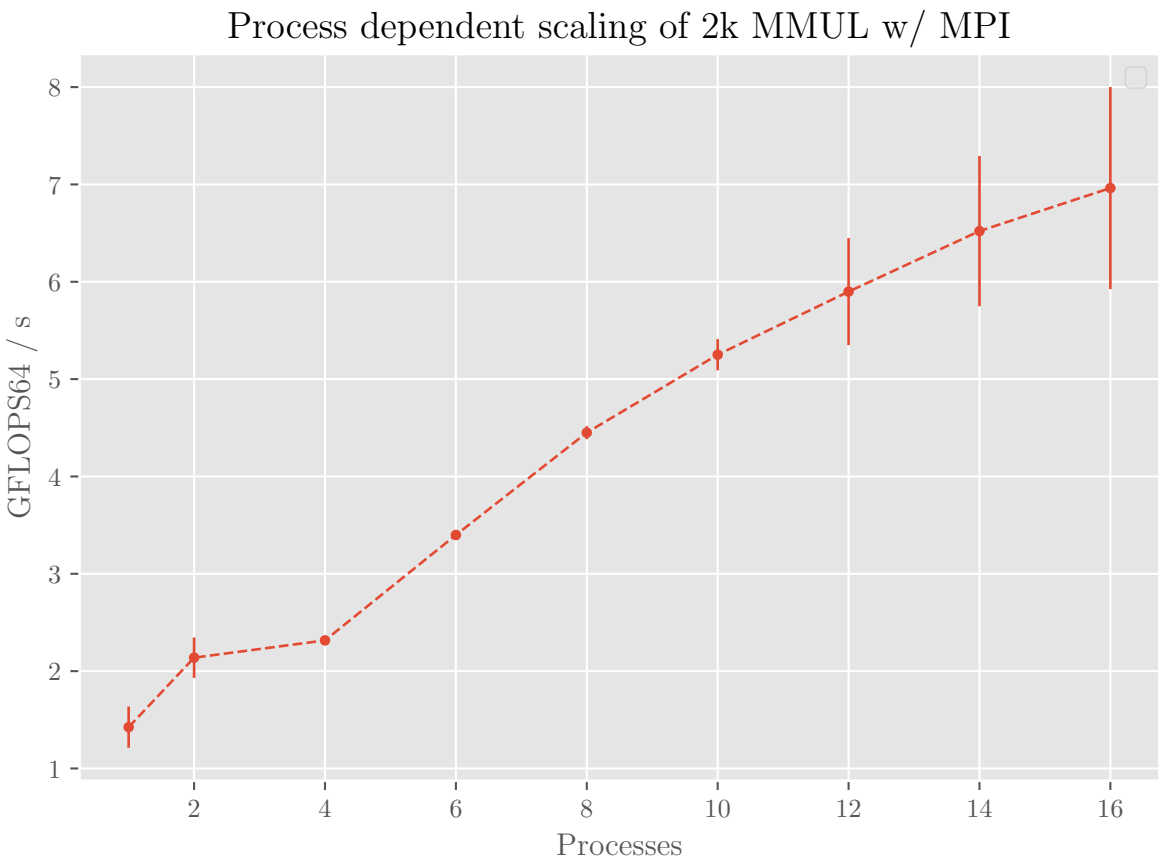

Figure 1: Scaling Process Count

4.4 Matrix multiply — scaling problem size

In addition to the process scaling in subsection 4.3, the scaling by problem size was also observed. The observed numbers are shown in Table 2, here each test was run for both 10 and 16 processes as well as repeated 10 times.

dim	nodes	GFLOPS64/s mean	std
128	10	0.778	0.058
	16	0.407	0.012
256	10	3.056	0.249
	16	2.523	0.106
512	10	5.164	0.204
	16	6.505	1.947
1024	10	4.529	0.687
	16	3.231	1.393
2048	10	5.281	0.132
	16	7.451	0.967
4096	10	5.593	0.054
	16	8.601	0.128

Table 2: Scaling by Problem Size in numbers

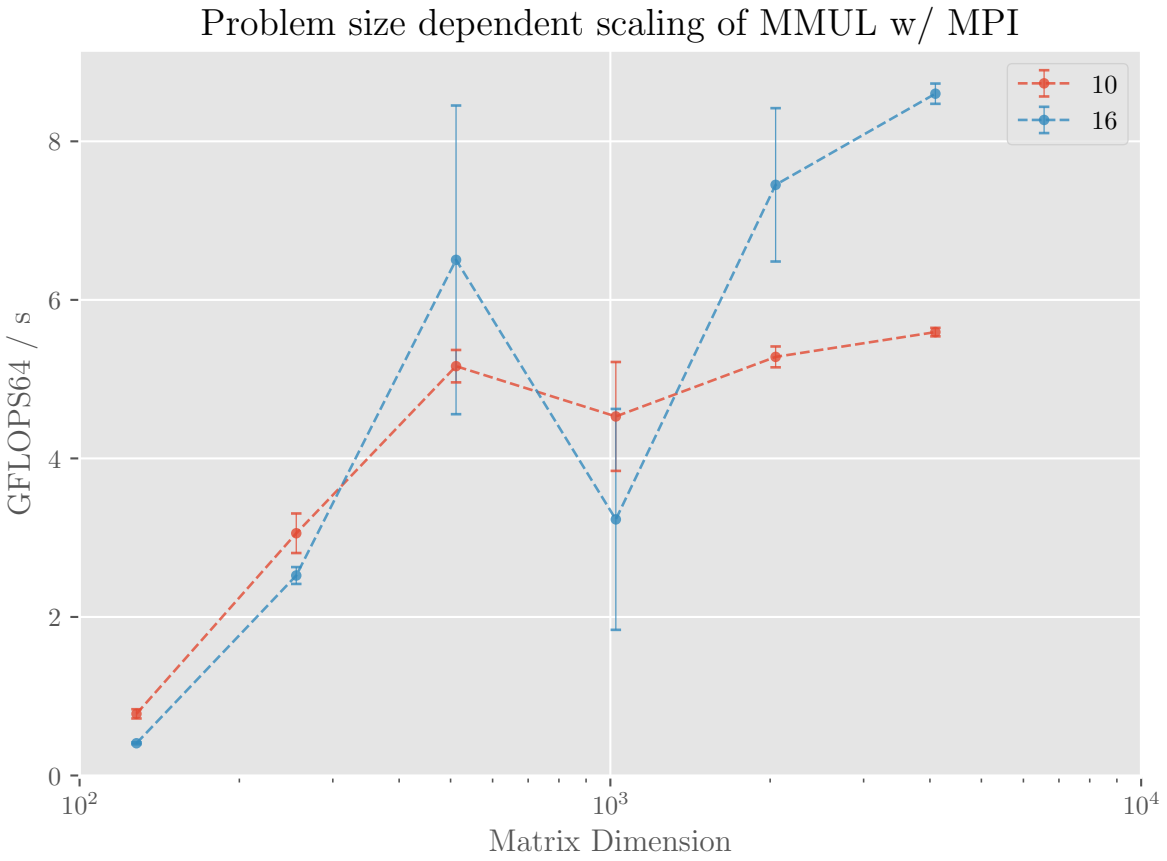

Figure 2: Scaling Problem Size