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

The Paper "The Future of Microprocessors" by Shekhar Borkar and Andrew Chien describes the end of performance scaling via transistor-speed of microprocessors. Instead, according to the authors, massive changes in architecture are necessary to keep up with Moore's Law. The main reason for this is energy efficiency which will become the key metric in chip design. By simply adding as many compute cores at maximum frequency as the transistor-integration capacity allows for the power consumption of the microprocessor would quickly grow into the hundreds of Watts. Instead of more fully fletched compute cores the authors suggest that instead both cache size will grow further since caches require less energy compared to logic and that microprocessors will instead feature smaller fine grained compute cores capable of operating at different frequencies or accelerators for special workloads.

Some of the predictions made in the paper held true. For example frequency adjustment for certain cores is common even in consumer grade CPUs (Intel Turbo Boost, AMD Turbo Core). On the other hand cache sizes did not grow as rapidly as predicted. Most off the shelf CPUs feature caches in the range of 30 MB with some going as high as 64 MB. Exceptions to this are exotic architectures like the *IBM z15* that feature very large off-chip caches realised as eDRAMs (960 MB L4 cache). Also the incorporation of smaller compute cores and accelerators into the CPU is mostly only found in exotic architectures. x64 based systems seem to stick to external many core accelerators like *GPUs* or the *Xeon Phi*. Despite that the core premise of the paper held true as energy efficiency has become a very important metric not only in HPC but also in consumer grade computing especially mobile devices. We therefore accept the paper.

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.

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.

nodes	GFLOPS64/s mean	std	t/ns mean	std	Speedup
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

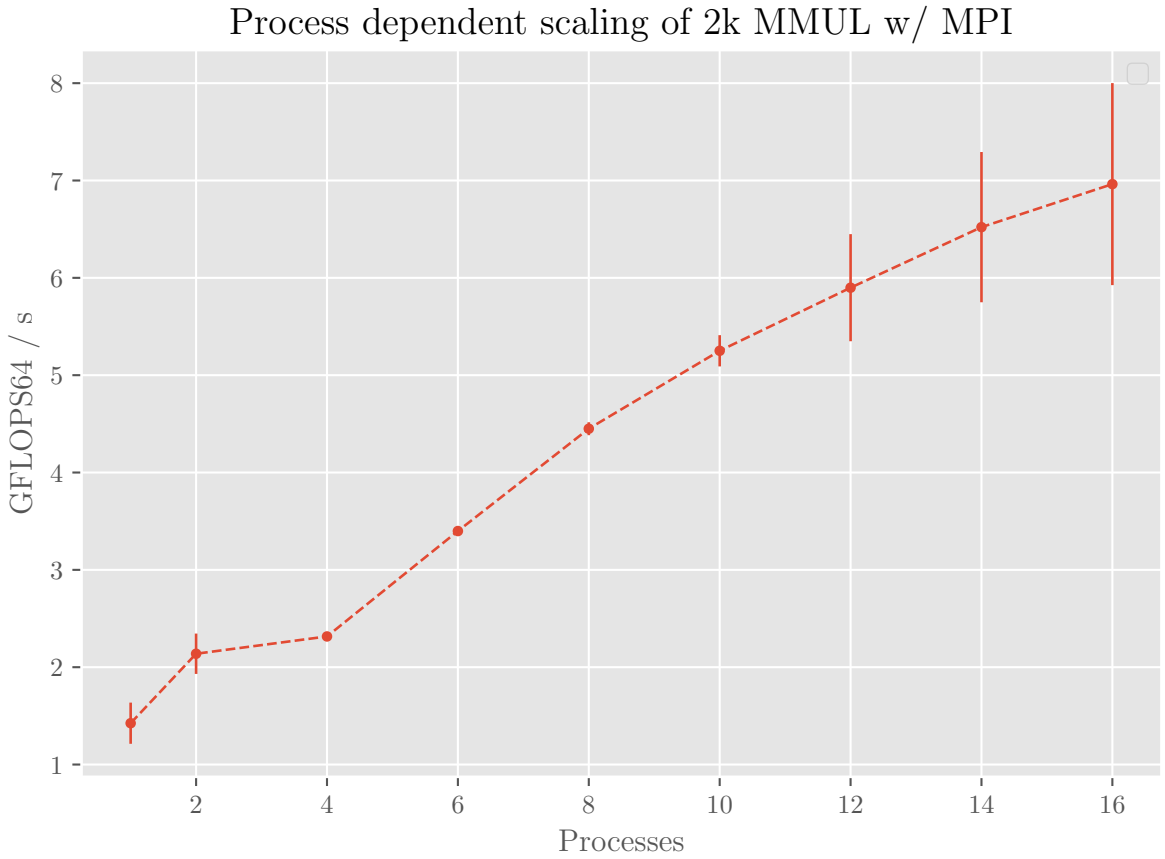

Figure 1: Scaling Process Count

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

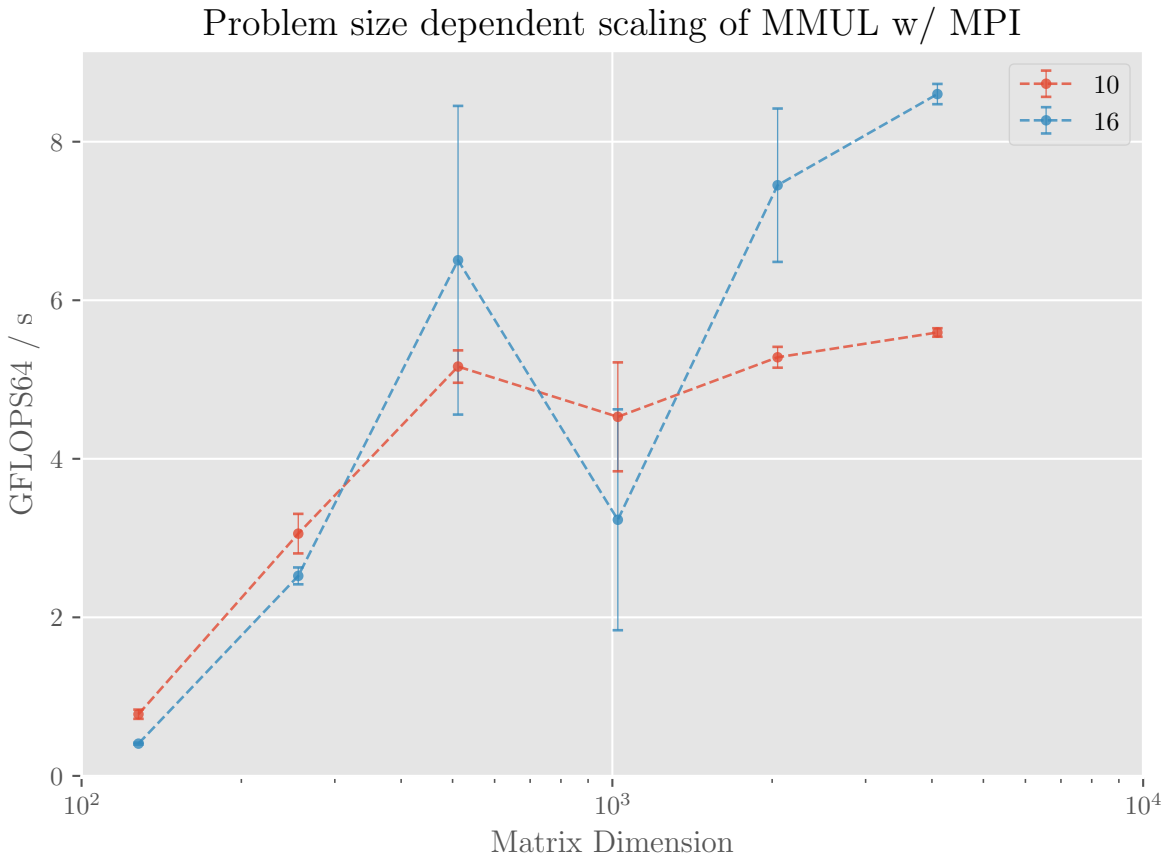

Figure 2: Scaling Problem Size