Exercise 3

3.1 Reading

The article „The Future of Microprocessors" by Shekhar Borkar and Andrew Chien, discusses the shift away from relying only on transistor speed for microprocessor performance scaling. The Authors argue, that to keep up with Moore's law, significant architectural changes are necessary, with a particular emphasis on prioritizing energy efficiency 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, the authors suggest, that cache sizes will increase, since they 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 predictions from the paper have become reality. For example, frequency adjustment for certain cores is common even in consumer grade CPUs (Intel Turbo Boost, AMD Turbo Core). However, cache sizes did not increase as rapidly as predicted. Most mainstream CPUs usually have a cache of around 30 MB, some reach 64 MB and therefore differ from the originally expected development. Notable exceptions, such as the IBM z15, use exotic architectures with very large off-chip caches, which are realised as eDRAMs and reach up to 960 MB in the L4 cache. The integration of smaller computing cores and accelerators in CPUs can be observed above all in exotic architectures, while x64-based systems often rely on external multi-core accelerators such as GPUs or the Xeon Phi.
Nevertheless, the core premise of the paper is correct, as energy efficiency has become a very important metric not only in the HPC sector, but also in the consumer sector, especially for mobile devices. We therefore accept the paper.

3.2 Matrix multiply – sequential version

The octance cluster has a AMD EPYC 7351P CPU, which according to Mandelbrot (https://browser.geekbench.com/v3/cpu/8555579) has a Single-core GFLOP/s performance of 3,35.

The naïve implementation uses three nested for loops to perform the matrix

It achieves the following results:
Running on one node
==========
Unoptimized :
258s 0.06639 GFlops/s
==========

There are two main problems with the implementation:

- Hardware utilization: The processor used has 16 CPU Cores. The naïve code, however, only uses one thread and core.

- Cache utilization: Cache utilization in the naive version is suboptimal due to a strided access pattern, causing cache misses for accesses to matrix B.

The optimised code is using a transposed matrix for matrix multiplication.
achieved performance GFLOP/s
This is achieved because multiple elements from the same cacheline can be used. It achieves the following results:

Running on one node

==========

Optimized :

40s 0.42928 GFlops/s

==========

Group HPC03:

3.2:

optimisation:
the optimised code is using a transposed matrix for matrix multiplication.
achieved performance GFLOP/s

This is achieved because multiple elements from the same cacheline can be used.

unoptimised:
runtime: 258s 0.06639 GFlops/s
optimized:
runtime: 40s 0.42928 GFlops/s

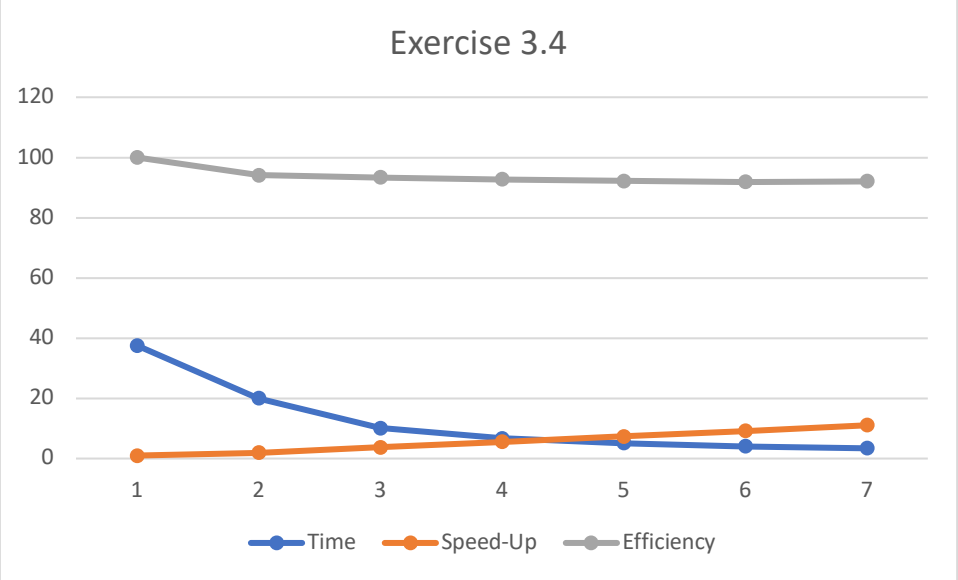
3.3:

C=[
0.000000 30.000000 60.000000 90.000000 120.000000
0.000000 40.000000 80.000000 120.000000 160.000000
0.000000 50.000000 100.000000 150.000000 200.000000
0.000000 60.000000 120.000000 180.000000 240.000000
0.000000 0.000000 0.000000 0.000000 0.000000]

We were not able to scatter the data unevenly between ranks, so in this case ⅘ lines of the output array are correct the performance for such a small matrix went down to 0.0005GFlops/s, with increasing size of the array the overhead introduced for Broadcasting and Gatter data is in relative smaller

Exercise 3.4

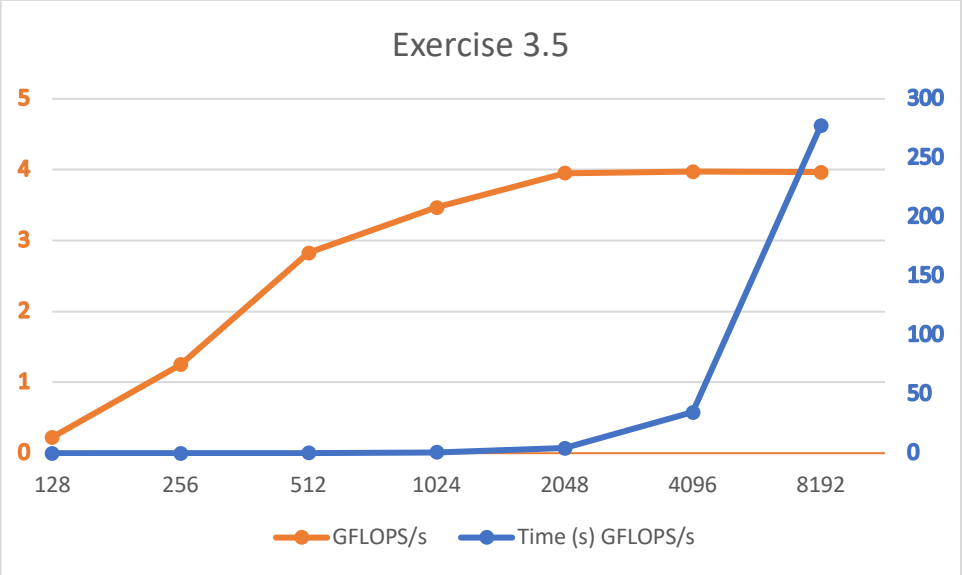
Process	Time	Speed-Up	Efficiency
1	37,56	1	100
2	19,95	1,88270677	94,1353383
4	10,06	3,73359841	93,3399602
6	6,75	5,56444444	92,7407407
8	5,09	7,37917485	92,2396857
10	4,09	9,18337408	91,8337408
12	3,4	11,0470588	92,0588235



As shown in the table there is a significant speed up to be achieved by parallelizing the matrix multiplication. However, it has to be noted that as the number of processes increases, the efficiency of parallelization tends to decrease. The fast increase of speed up is also illustrated in the diagram above (blue line).

Exercise 3.5

Size	Time (s)	GFLOPS/s
128	0,018594	0,225578
256	0,026827	1,25
512	0,094914	2,8282
1024	0,619426	3,466891
2048	4,348465	3,95079
4096	34,584087	3,974052
8192	277,430282	3,9632



As the matrix size increases, the time taken for the multiplication operation also increases exponentially, as expected due to the computational complexity. The performance, measured in GFLOPS/s increases for larger matrix sizes until it reaches a stagnation at 4 GFLOPS/s.

3.6 Willingness to present

Willing to present all exercises.