For further improvements over the OpenMP implementation we have used the AVX extension, which supports simultaneous 4 operations on double numbers (64 bit floating point numbers) per CPU core. The application to be benchmarked is the matrix-matrix multiplication as requested at exercise 1a.3).

The theoretical improvement should be 4 over the OpenMP configuration (in our case, 8 threads achieved highest performance, running on a 4-core/8 threads Intel CPU). However, from the graph below it can be seen that the maximum speed-up is slightly above 2 over the parallel implementation and aprox. 6.3 over the sequential one.

The implementation details and the code are provided in the updated matrix3.c file. The application was running on an Intel i7-4700MQ quad-core mobile (laptop) CPU, with tuned frequency at 2.5 Ghz for any core count (we had to bypass the Intel TurboBoost technology in order to get relevant results) and 8 GB of dual-channel DDR3 RAM (2x4GB).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Matrix size (N) | Sequential (sec) | Parallel (sec) | AVX + FMA (sec) | Speed-up over OpenMP | Speed-up over sequential |
| 384 | 0.522906 | 0.180749 | 0.097184 | 1.859864 | 5.380577 |
| 480 | 0.84631 | 0.277155 | 0.136248 | 2.034195 | 6.211541 |
| 512 | 1.285152 | 0.453968 | 0.204369 | 2.221315 | 6.28839 |
| 640 | 2.499389 | 0.828006 | 0.610611 | 1.356029 | 4.093259 |
| 720 | 2.981393 | 0.970554 | 0.735541 | 1.31951 | 4.053334 |
| 800 | 4.494748 | 1.402903 | 1.141197 | 1.229326 | 3.938626 |
| 1024 | 17.60882 | 4.447762 | 3.374153 | 1.318186 | 5.218737 |
| 2048 | 217.2149 | 64.9183 | 48.79517 | 1.330425 | 4.451566 |

Table. Results in seconds for running different size (N) matrix x matrix multiplication in the three mentioned implementations. Speed-up results are shown in the two left-most columns.

Chart. Computed speed-ups where the reference is the AVX with FMA implementation over parallel and sequential code