Results

While the expected results have also been placed in the “test\_case\_constants.h” file, I have also put two of the results here as examples. The program currently prints out the two input matrices before the result. I hope this does not interfere too much with the marking of my solution.

The time taken to run my program is slower than the time needed for single threaded calculators to finish, which is likely due to the overhead associated with creating new processes, the blocking of the earlier children instead of allowing each one to start when created, significant numbers of function calls, as well as the extra steps used to load the matrices into the shared memory.

|  |  |  |  |
| --- | --- | --- | --- |
| Provided test case (MN) | | | |
| 990 | 1150 | 1530 | 1990 |
| 800 | 716 | 918 | 1254 |
| 501 | 1279 | 2068 | 2858 |
| 181 | 321 | 492 | 668 |

|  |  |  |  |
| --- | --- | --- | --- |
| First extra test case | | | |
| 2 | -107 | -13 | 9 |
| -12 | -294 | -58 | 162 |
| 8 | -428 | -52 | 36 |
| 16 | 1320 | 152 | -312 |

Design Goals

The primary goal of my design (besides experimenting with taught material) was to try to make a framework which could work for any sort of matrix multiplication, after which I would make the matrix size constant for the deliverable. To follow this guideline, functions that directly interact with elements in a matrix are in “row\_calc.c”, and matrices are treated as variables and passed by reference. These two design choices likely made my program more convoluted and likely more resource heavy, but it should be possible to use a bit of work to refactor the program for multiplying matrices of any size or even for other matrix operations.