
University of California, Riverside
Department of Computer Science and Engineering

# Title: Programming Assignment 2

## Course: Multiprocessor Architecture and Programming

Mahbod Afarin

Winter 2021

The goal of this project is to improve the performance of the sparse matrix-vector multiplication using OpenMP. The is 4 implementations in the project folder, CSRSeq.c, CSROpenMP.c, CSCSeq.c, and CSCOpenMP.c. You can run each code using the below command. For running the program, please use the input files in the project because, as it was allowed in piazza, I changed the format of the input files from .rtf format to .text format.

$$/usr/local/opt/llvm/bin/clang\ -fopenmp\ -L/usr/local/opt/llvm/lib\ CSRSeq.c\ -o\ smpv$$

In the figure 1 you can see a sample output of the program for 8 iterations and 32 threads for CSROpenMP.c.

```
mahbodafarin1admin@Rajivs-MacBook-Air code % ./smpv matrix3.rtf 8 32
Loading the input matrix "matrix3.rtf"
Computing the results for 8 iterations (Parallel Version)...
Number of threads = 32
The execution time is: 15399.932861 us
```

*Figure 1: Screenshot of the output*

Table 1 shows the execution time and corresponding speedup for different number of threads for different matrices for CSR and CSR serial and parallel implementation. I did the multithreading optimization when the program wants to do the computation. For the matrix1, the multithreading makes the execution time worser than the serial version for both CSR and CSC because the input matrix is already so small and making it parallel add it some overheads for context switching between threads, communication because between threads, memory bound, and I/O bound. For the matrix2, the execution time is better for parallel version compared to the serial version of the program for both CSR and CSC because the input matrix is bigger and the benefits of parallel version is bigger than its overheads, but increasing the number of threads makes the execution time worser because the program dose not have infinity potential of parallelism and when we reach that point, assigning more threads just add more overheads and does not help it to improve the parallelism. For the matrix3, we have a very good speedup because the size of matrix is bigger, so the potential of parallelism is better, but it has the problems of matrix2 which I already mentioned.

*Table 1: Execution Time and speedup for sequential version and parallel version with different number of threads*

| Number of Threads | Run Time for CSR (Micro Seconds) | | | Run Time for CSC (Micro Seconds) | | | Speedup for CSR | | | Speedup for CSC | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Matrix1 | Matrix2 | Matrix3 | Matrix1 | Matrix2 | Matrix3 | M1 | M2 | M3 | M1 | M2 | M3 |
| **Sequential** | 458 | 5556 | 34099 | 1207 | 5021 | 30821 | - | - | - | - | - | - |
| **2** | 3240 | 3640 | 14219 | 2815 | 2861 | 17382 | 0.14 | 1.53 | 2.4 | 0.43 | 1.75 | 1.77 |
| **3** | 2547 | 3917 | 13473 | 2741 | 2584 | 16482 | 0.18 | 1.42 | 2.53 | 0.44 | 1.94 | 1.87 |
| **4** | 3064 | 3547 | 11920 | 2354 | 2386 | 17347 | 0.15 | 1.57 | 2.86 | 0.51 | 2.1 | 1.78 |
| **5** | 2835 | 3465 | 13518 | 2788 | 2751 | 17183 | 0.16 | 1.6 | 2.52 | 0.43 | 1.83 | 1.79 |
| **6** | 2830 | 3761 | 13568 | 2157 | 2779 | 15831 | 0.16 | 1.48 | 2.51 | 0.56 | 1.81 | 1.95 |
| **7** | 3309 | 4292 | 12591 | 2934 | 2912 | 17742 | 0.14 | 1.29 | 2.71 | 0.41 | 1.72 | 1.74 |
| **8** | 2547 | 3731 | 12907 | 2458 | 3591 | 16196 | 0.18 | 1.49 | 2.64 | 0.49 | 1.4 | 1.9 |
| **9** | 4881 | 3278 | 12211 | 3546 | 3416 | 15628 | 0.09 | 1.69 | 2.79 | 0.34 | 1.47 | 1.97 |
| **10** | 4277 | 3479 | 12985 | 2411 | 3785 | 17527 | 0.11 | 1.6 | 2.63 | 0.5 | 1.33 | 1.76 |
| **11** | 3411 | 3674 | 12747 | 2875 | 4121 | 14042 | 0.13 | 1.51 | 2.68 | 0.42 | 1.22 | 2.19 |
| **12** | 4300 | 3764 | 12332 | 3121 | 3261 | 15637 | 0.11 | 1.48 | 2.77 | 0.39 | 1.54 | 1.97 |
| **13** | 1175 | 4227 | 12748 | 3588 | 3659 | 14431 | 0.39 | 1.31 | 2.67 | 0.34 | 1.37 | 2.14 |
| **14** | 3331 | 3583 | 13545 | 3182 | 2581 | 13784 | 0.14 | 1.55 | 2.52 | 0.38 | 1.95 | 2.24 |
| **15** | 3574 | 5943 | 14178 | 3881 | 3410 | 15345 | 0.13 | 0.93 | 2.41 | 0.31 | 1.47 | 2.01 |
| **16** | 4352 | 7460 | 12665 | 3714 | 3541 | 16689 | 0.11 | 0.74 | 2.69 | 0.32 | 1.42 | 1.85 |
| **17** | 5023 | 3341 | 14588 | 3818 | 3861 | 12128 | 0.09 | 1.66 | 2.34 | 0.32 | 1.3 | 2.54 |
| **18** | 3587 | 6551 | 12752 | 3715 | 3517 | 14432 | 0.13 | 0.85 | 2.67 | 0.32 | 1.43 | 2.14 |
| **19** | 3248 | 5778 | 13741 | 3912 | 3841 | 1373 | 0.14 | 0.96 | 2.48 | 0.31 | 1.31 | 22.45 |
| **20** | 5707 | 6536 | 13027 | 4018 | 2914 | 1597 | 0.08 | 0.85 | 2.62 | 0.3 | 1.72 | 19.3 |
| **21** | 2042 | 4752 | 13301 | 4715 | 4031 | 13432 | 0.22 | 1.17 | 2.56 | 0.26 | 1.25 | 2.29 |
| **22** | 3675 | 4510 | 14060 | 4193 | 4218 | 14763 | 0.12 | 1.23 | 2.43 | 0.29 | 1.19 | 2.09 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **23** | 3548 | 4446 | 12789 | 4701 | 3821 | 15243 | 0.13 | 1.25 | 2.67 | 0.26 | 1.31 | 2.02 |
| **24** | 2271 | 3889 | 13041 | 4991 | 4817 | 13472 | 0.2 | 1.43 | 2.61 | 0.24 | 1.04 | 2.29 |
| **25** | 2155 | 3251 | 15869 | 4820 | 4681 | 12179 | 0.21 | 1.71 | 2.15 | 0.25 | 1.07 | 2.53 |
| **26** | 2109 | 4145 | 13085 | 4301 | 4871 | 15432 | 0.22 | 1.34 | 2.61 | 0.28 | 1.03 | 2 |
| **27** | 2146 | 4997 | 14906 | 4857 | 4512 | 13127 | 0.21 | 1.11 | 2.29 | 0.25 | 1.11 | 2.35 |
| **28** | 2346 | 4693 | 15407 | 5028 | 4871 | 12315 | 0.2 | 1.18 | 2.21 | 0.24 | 1.03 | 2.5 |
| **29** | 3685 | 4896 | 16379 | 4810 | 4211 | 14472 | 0.12 | 1.13 | 2.08 | 0.25 | 1.19 | 2.13 |
| **30** | 3805 | 4384 | 15037 | 4217 | 4723 | 13167 | 0.12 | 1.27 | 2.27 | 0.29 | 1.06 | 2.34 |
| **31** | 2341 | 4773 | 14489 | 4923 | 4871 | 12128 | 0.2 | 1.16 | 2.35 | 0.25 | 1.03 | 2.54 |
| **32** | 3447 | 4342 | 15363 | 4821 | 4578 | 15361 | 0.13 | 1.28 | 2.22 | 0.25 | 1.1 | 2.01 |