Spring 2022: Advanced Topics in Numerical Analysis: High Performance Computing Assignment 2 (due Mar. 21, 2022)

Handing in your homework: Please create a Git repository on Github to hand in your homework. If you choose your repository to be private, please give Melody and Cai (who are helping with grading) read access to the repo (Melody's username is `melodyshih` and Cai's is `caim-d`). The repository should contain the source files, as well as a Makefile. To hand in your homework, please email me and Melody together a single message with the location of your repo. Generate a `hw2` directory in this repo, which contains all the source code and a short `.txt` or LaTeX file that answers the questions/reports timings from this assignment. Alternatively, you can hand in a sheet with the answers/timings that also specifies the location of your repo. To check if your code runs, we will type the following commands[1]:

```
git clone YOURPATH/YOURREPO.git
cd YOURREPO/hw2/
make
./MMult1
./val_test01_solved
./val_test02_solved
./omp_solved2
...
./omp_solved6
./jacobi2D-omp
./gs2D-omp
```

The folder homework02 in the git repository https://github.com/pehersto/HPCSpring2022 contains the matrix-matrix multiplication code you can start with, and the buggy code examples needed for this homework.[2] For an overview over OpenMP, you can use the material and examples from class and the official documentation for the OpenMP Standard 5.0.[3]

1. **Finding Memory bugs.** The homework repository contains two simple programs that contain bugs. Use valgrind to find these bugs and fix them. Add a short comment to the code describing what was wrong and how you fixed the problem. Add the solutions to your repository using the naming convention `val_test01_solved.cpp`, `val_test02_solved.cpp`, and use the Makefile to compile the example problems.
   Solutions:
   Fixed the code and added comments in the code itself

[1] Since we will partially automize this, make sure this will work for the code you hand in.

[2] And yes, these examples can also be found on the web—but for your own sake, please try to understand what's going on and try to find the bugs.

[3] http://www.openmp.org/specifications/

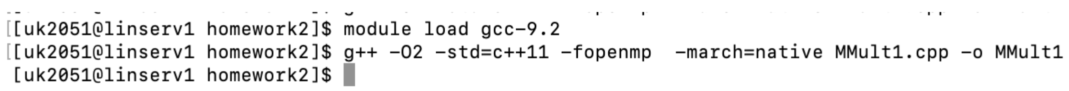
2. **Optimizing matrix-matrix multiplication.** In this homework you will optimize the matrix-matrix multiplication code from the last homework using blocking. This increases the computational intensity (i.e., the ratio of flops per access to the slow memory) and thus speed up the implementation substantially. The code you can start with, along with further instructions are in the source file `MMult1.cpp`. Specifying what machine you run on, hand in timings for various matrix sizes obtained with the blocked version and the OpenMP version of the code.

   Solutions:

   NOTE: Please do module load gcc-9.2, otherwise there will be trouble compiling it

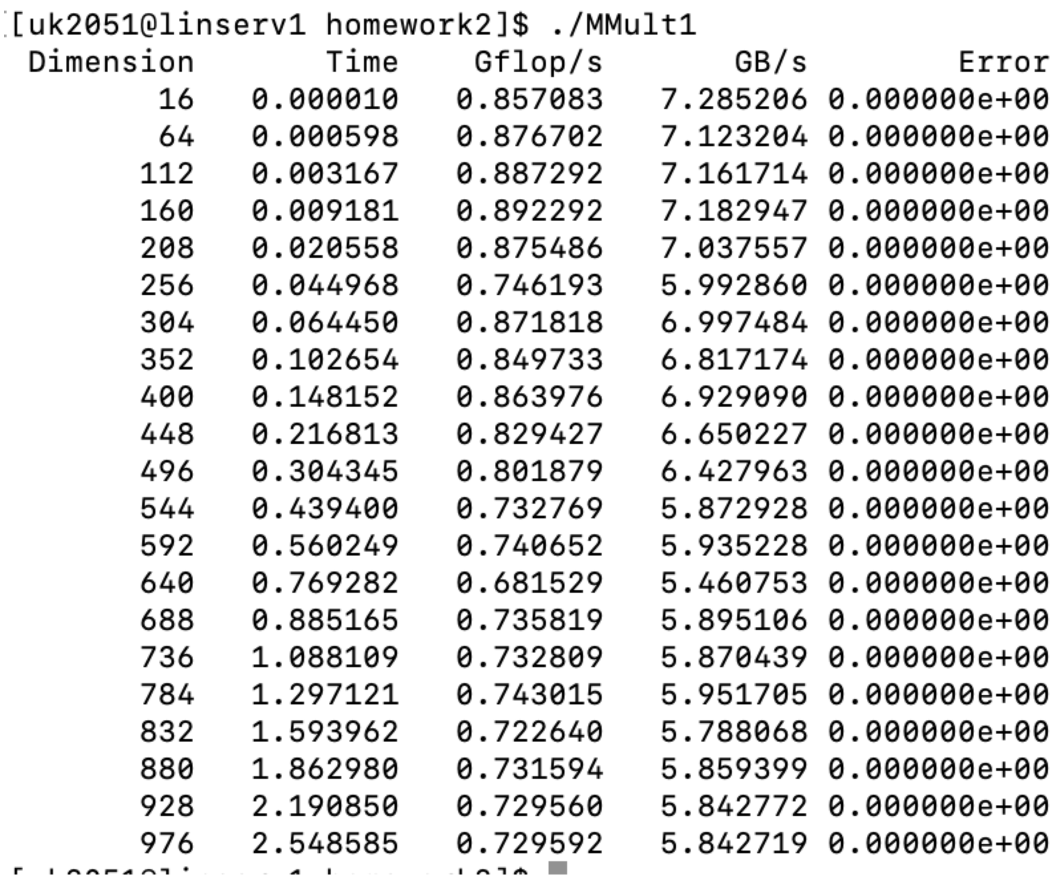
   Machine Description: https://linserv1.cims.nyu.edu

   Following is a screenshot depicting how to do it:

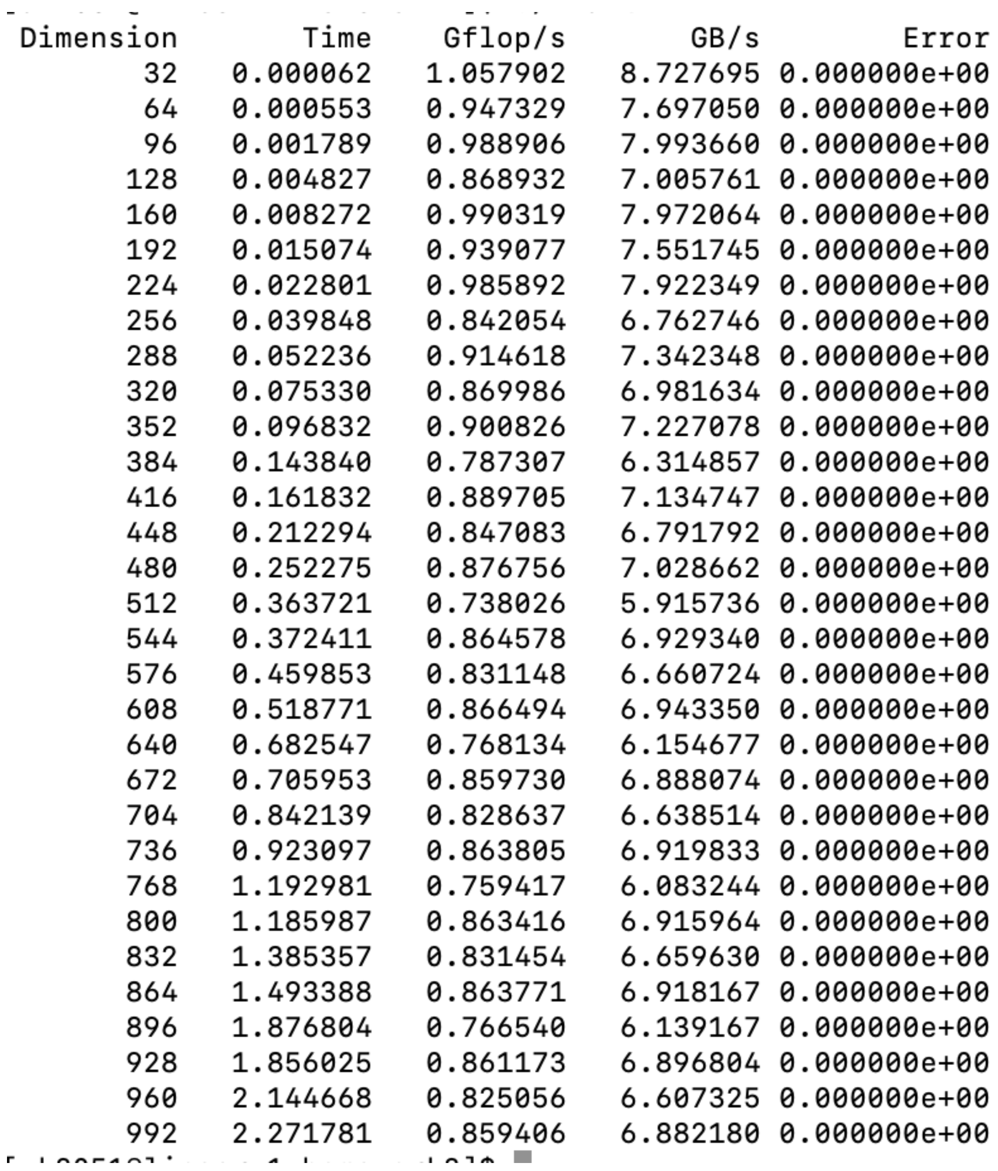


```
[[uk2051@linserv1 homework2]$ module load gcc-9.2
[[uk2051@linserv1 homework2]$ g++ -O2 -std=c++11 -fopenmp  -march=native MMult1.cpp -o MMult1
 [uk2051@linserv1 homework2]$
```

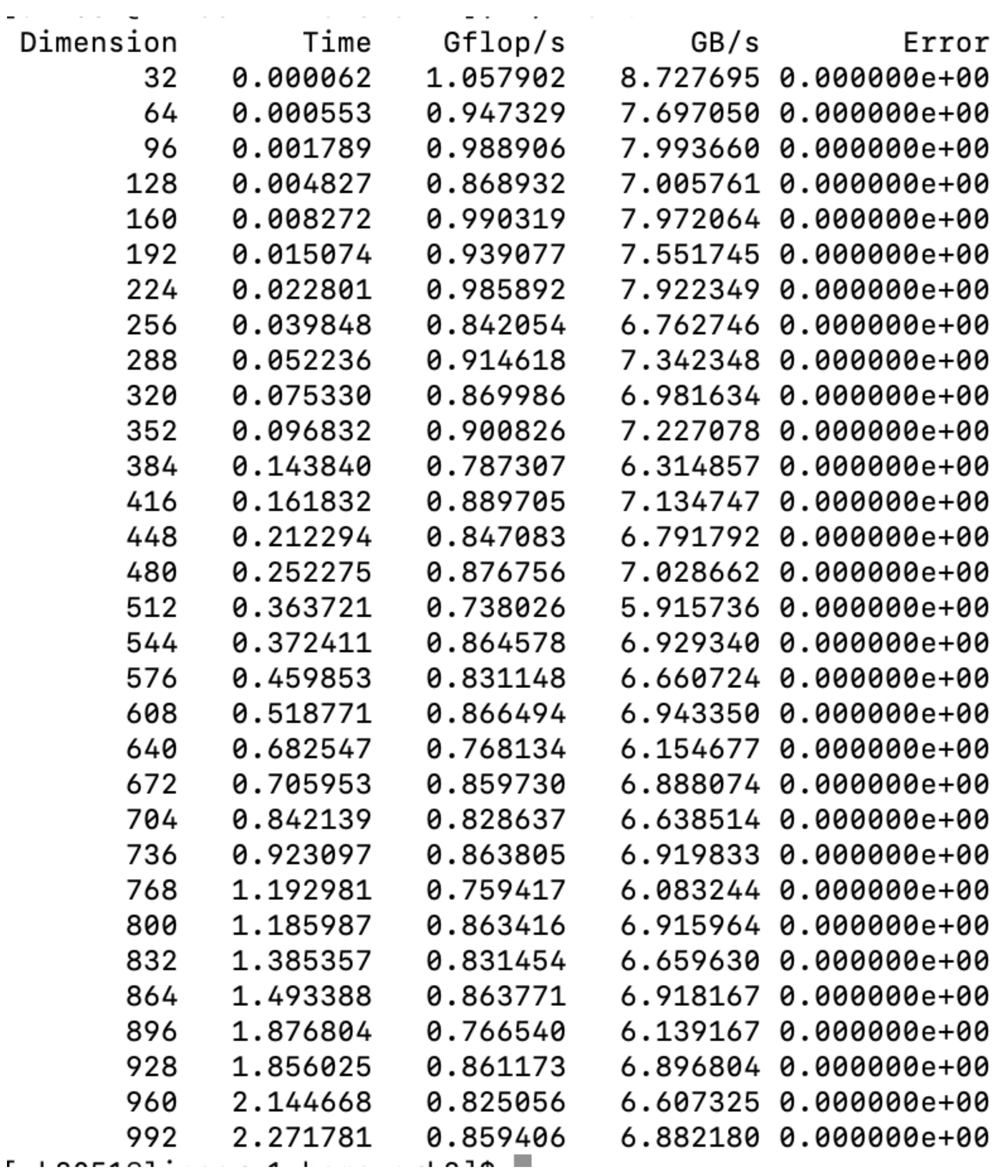
   Below is the screenshot for various matrices size using the blocked code without parallelization:



```
[uk2051@linserv1 homework2]$ ./MMult1
 Dimension       Time    Gflop/s       GB/s        Error
        16   0.000010   0.857083   7.285206 0.000000e+00
        64   0.000598   0.876702   7.123204 0.000000e+00
       112   0.003167   0.887292   7.161714 0.000000e+00
       160   0.009181   0.892292   7.182947 0.000000e+00
       208   0.020558   0.875486   7.037557 0.000000e+00
       256   0.044968   0.746193   5.992860 0.000000e+00
       304   0.064450   0.871818   6.997484 0.000000e+00
       352   0.102654   0.849733   6.817174 0.000000e+00
       400   0.148152   0.863976   6.929090 0.000000e+00
       448   0.216813   0.829427   6.650227 0.000000e+00
       496   0.304345   0.801879   6.427963 0.000000e+00
       544   0.439400   0.732769   5.872928 0.000000e+00
       592   0.560249   0.740652   5.935228 0.000000e+00
       640   0.769282   0.681529   5.460753 0.000000e+00
       688   0.885165   0.735819   5.895106 0.000000e+00
       736   1.088109   0.732809   5.870439 0.000000e+00
       784   1.297121   0.743015   5.951705 0.000000e+00
       832   1.593962   0.722640   5.788068 0.000000e+00
       880   1.862980   0.731594   5.859399 0.000000e+00
       928   2.190850   0.729560   5.842772 0.000000e+00
       976   2.548585   0.729592   5.842719 0.000000e+00
```

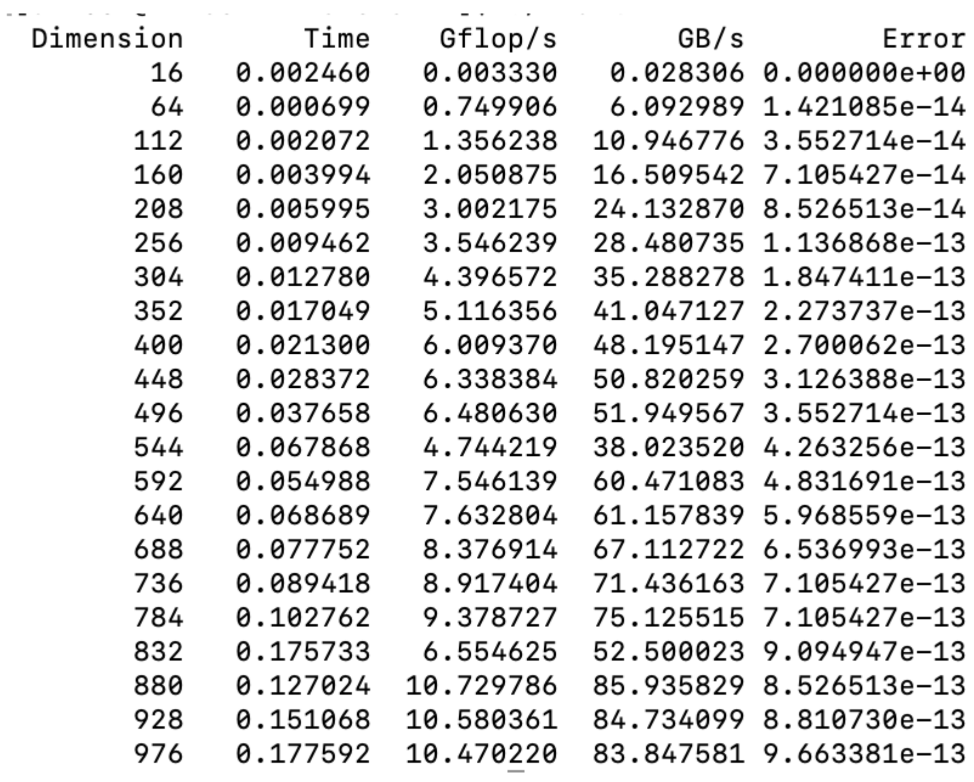


```
 Dimension        Time     Gflop/s        GB/s        Error
        32    0.000062    1.057902    8.727695 0.000000e+00
        64    0.000553    0.947329    7.697050 0.000000e+00
        96    0.001789    0.988906    7.993660 0.000000e+00
       128    0.004827    0.868932    7.005761 0.000000e+00
       160    0.008272    0.990319    7.972064 0.000000e+00
       192    0.015074    0.939077    7.551745 0.000000e+00
       224    0.022801    0.985892    7.922349 0.000000e+00
       256    0.039848    0.842054    6.762746 0.000000e+00
       288    0.052236    0.914618    7.342348 0.000000e+00
       320    0.075330    0.869986    6.981634 0.000000e+00
       352    0.096832    0.900826    7.227078 0.000000e+00
       384    0.143840    0.787307    6.314857 0.000000e+00
       416    0.161832    0.889705    7.134747 0.000000e+00
       448    0.212294    0.847083    6.791792 0.000000e+00
       480    0.252275    0.876756    7.028662 0.000000e+00
       512    0.363721    0.738026    5.915736 0.000000e+00
       544    0.372411    0.864578    6.929340 0.000000e+00
       576    0.459853    0.831148    6.660724 0.000000e+00
       608    0.518771    0.866494    6.943350 0.000000e+00
       640    0.682547    0.768134    6.154677 0.000000e+00
       672    0.705953    0.859730    6.888074 0.000000e+00
       704    0.842139    0.828637    6.638514 0.000000e+00
       736    0.923097    0.863805    6.919833 0.000000e+00
       768    1.192981    0.759417    6.083244 0.000000e+00
       800    1.185987    0.863416    6.915964 0.000000e+00
       832    1.385357    0.831454    6.659630 0.000000e+00
       864    1.493388    0.863771    6.918167 0.000000e+00
       896    1.876804    0.766540    6.139167 0.000000e+00
       928    1.856025    0.861173    6.896804 0.000000e+00
       960    2.144668    0.825056    6.607325 0.000000e+00
       992    2.271781    0.859406    6.882180 0.000000e+00
```

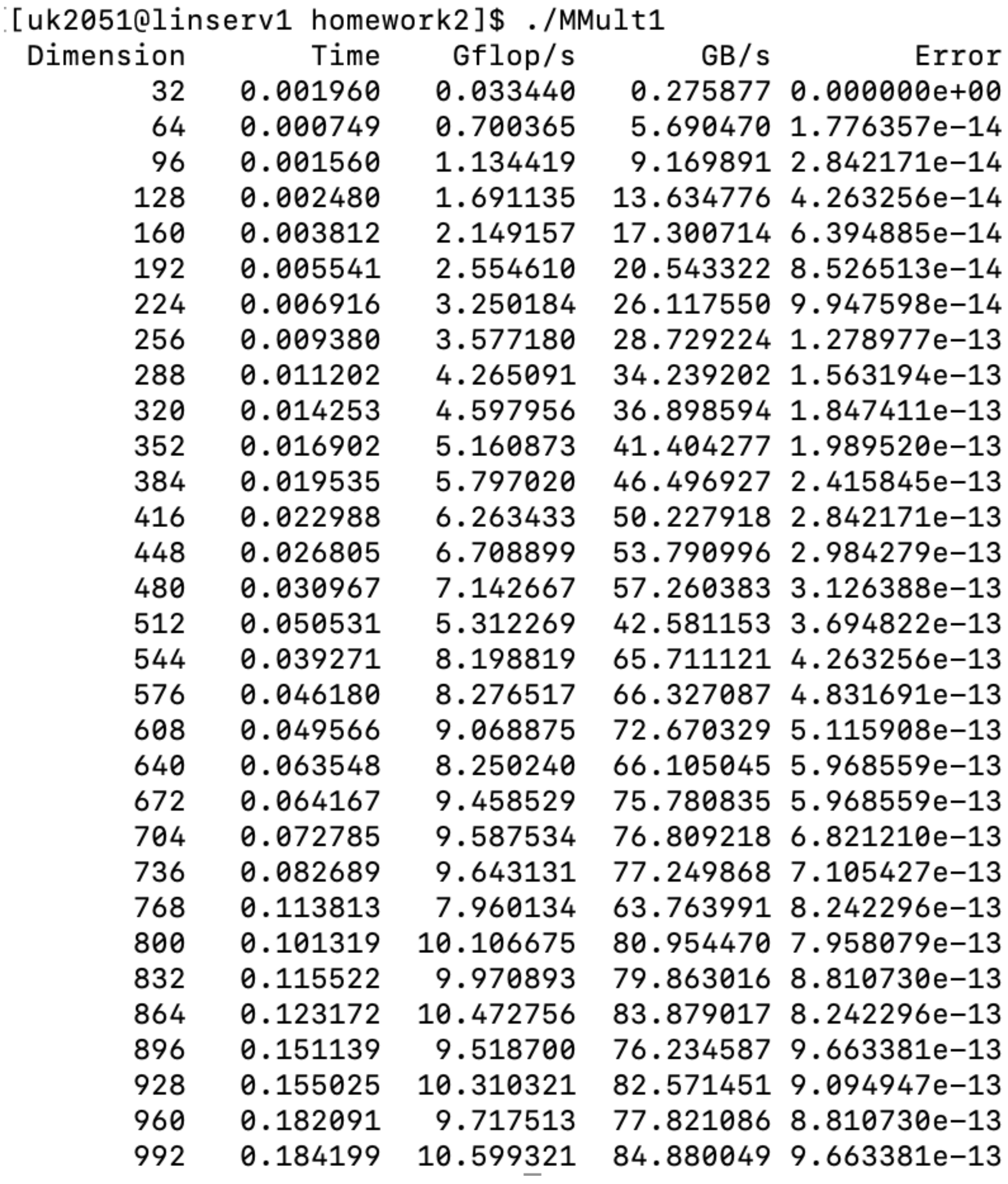


```
 Dimension       Time    Gflop/s       GB/s        Error
        32   0.000062   1.057902   8.727695 0.000000e+00
        64   0.000553   0.947329   7.697050 0.000000e+00
        96   0.001789   0.988906   7.993660 0.000000e+00
       128   0.004827   0.868932   7.005761 0.000000e+00
       160   0.008272   0.990319   7.972064 0.000000e+00
       192   0.015074   0.939077   7.551745 0.000000e+00
       224   0.022801   0.985892   7.922349 0.000000e+00
       256   0.039848   0.842054   6.762746 0.000000e+00
       288   0.052236   0.914618   7.342348 0.000000e+00
       320   0.075330   0.869986   6.981634 0.000000e+00
       352   0.096832   0.900826   7.227078 0.000000e+00
       384   0.143840   0.787307   6.314857 0.000000e+00
       416   0.161832   0.889705   7.134747 0.000000e+00
       448   0.212294   0.847083   6.791792 0.000000e+00
       480   0.252275   0.876756   7.028662 0.000000e+00
       512   0.363721   0.738026   5.915736 0.000000e+00
       544   0.372411   0.864578   6.929340 0.000000e+00
       576   0.459853   0.831148   6.660724 0.000000e+00
       608   0.518771   0.866494   6.943350 0.000000e+00
       640   0.682547   0.768134   6.154677 0.000000e+00
       672   0.705953   0.859730   6.888074 0.000000e+00
       704   0.842139   0.828637   6.638514 0.000000e+00
       736   0.923097   0.863805   6.919833 0.000000e+00
       768   1.192981   0.759417   6.083244 0.000000e+00
       800   1.185987   0.863416   6.915964 0.000000e+00
       832   1.385357   0.831454   6.659630 0.000000e+00
       864   1.493388   0.863771   6.918167 0.000000e+00
       896   1.876804   0.766540   6.139167 0.000000e+00
       928   1.856025   0.861173   6.896804 0.000000e+00
       960   2.144668   0.825056   6.607325 0.000000e+00
       992   2.271781   0.859406   6.882180 0.000000e+00
```

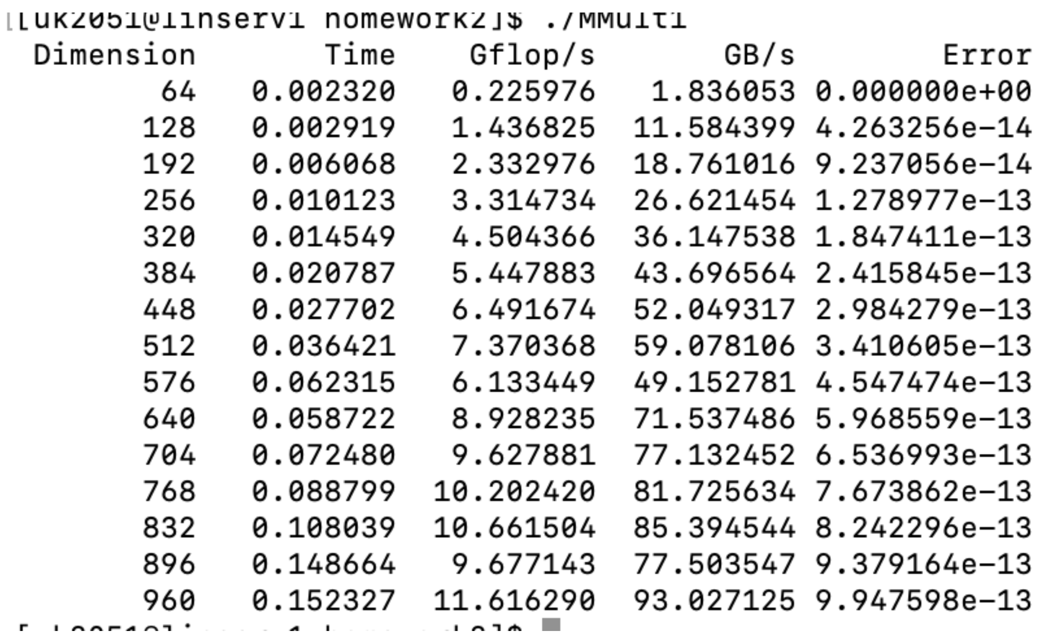
Below is the screenshot for various matrices size using the blocked code with parallelization using OpenMP:



```
 Dimension        Time    Gflop/s       GB/s        Error
        16    0.002460   0.003330   0.028306 0.000000e+00
        64    0.000699   0.749906   6.092989 1.421085e-14
       112    0.002072   1.356238  10.946776 3.552714e-14
       160    0.003994   2.050875  16.509542 7.105427e-14
       208    0.005995   3.002175  24.132870 8.526513e-14
       256    0.009462   3.546239  28.480735 1.136868e-13
       304    0.012780   4.396572  35.288278 1.847411e-13
       352    0.017049   5.116356  41.047127 2.273737e-13
       400    0.021300   6.009370  48.195147 2.700062e-13
       448    0.028372   6.338384  50.820259 3.126388e-13
       496    0.037658   6.480630  51.949567 3.552714e-13
       544    0.067868   4.744219  38.023520 4.263256e-13
       592    0.054988   7.546139  60.471083 4.831691e-13
       640    0.068689   7.632804  61.157839 5.968559e-13
       688    0.077752   8.376914  67.112722 6.536993e-13
       736    0.089418   8.917404  71.436163 7.105427e-13
       784    0.102762   9.378727  75.125515 7.105427e-13
       832    0.175733   6.554625  52.500023 9.094947e-13
       880    0.127024  10.729786  85.935829 8.526513e-13
       928    0.151068  10.580361  84.734099 8.810730e-13
       976    0.177592  10.470220  83.847581 9.663381e-13
```



```
[uk2051@linserv1 homework2]$ ./MMult1
 Dimension       Time    Gflop/s       GB/s        Error
        32   0.001960   0.033440   0.275877 0.000000e+00
        64   0.000749   0.700365   5.690470 1.776357e-14
        96   0.001560   1.134419   9.169891 2.842171e-14
       128   0.002480   1.691135  13.634776 4.263256e-14
       160   0.003812   2.149157  17.300714 6.394885e-14
       192   0.005541   2.554610  20.543322 8.526513e-14
       224   0.006916   3.250184  26.117550 9.947598e-14
       256   0.009380   3.577180  28.729224 1.278977e-13
       288   0.011202   4.265091  34.239202 1.563194e-13
       320   0.014253   4.597956  36.898594 1.847411e-13
       352   0.016902   5.160873  41.404277 1.989520e-13
       384   0.019535   5.797020  46.496927 2.415845e-13
       416   0.022988   6.263433  50.227918 2.842171e-13
       448   0.026805   6.708899  53.790996 2.984279e-13
       480   0.030967   7.142667  57.260383 3.126388e-13
       512   0.050531   5.312269  42.581153 3.694822e-13
       544   0.039271   8.198819  65.711121 4.263256e-13
       576   0.046180   8.276517  66.327087 4.831691e-13
       608   0.049566   9.068875  72.670329 5.115908e-13
       640   0.063548   8.250240  66.105045 5.968559e-13
       672   0.064167   9.458529  75.780835 5.968559e-13
       704   0.072785   9.587534  76.809218 6.821210e-13
       736   0.082689   9.643131  77.249868 7.105427e-13
       768   0.113813   7.960134  63.763991 8.242296e-13
       800   0.101319  10.106675  80.954470 7.958079e-13
       832   0.115522   9.970893  79.863016 8.810730e-13
       864   0.123172  10.472756  83.879017 8.242296e-13
       896   0.151139   9.518700  76.234587 9.663381e-13
       928   0.155025  10.310321  82.571451 9.094947e-13
       960   0.182091   9.717513  77.821086 8.810730e-13
       992   0.184199  10.599321  84.880049 9.663381e-13
```



```
[[uk2051@linserv1 homework2]$ ./MMult1
 Dimension       Time    Gflop/s       GB/s        Error
        64   0.002320   0.225976   1.836053 0.000000e+00
       128   0.002919   1.436825  11.584399 4.263256e-14
       192   0.006068   2.332976  18.761016 9.237056e-14
       256   0.010123   3.314734  26.621454 1.278977e-13
       320   0.014549   4.504366  36.147538 1.847411e-13
       384   0.020787   5.447883  43.696564 2.415845e-13
       448   0.027702   6.491674  52.049317 2.984279e-13
       512   0.036421   7.370368  59.078106 3.410605e-13
       576   0.062315   6.133449  49.152781 4.547474e-13
       640   0.058722   8.928235  71.537486 5.968559e-13
       704   0.072480   9.627881  77.132452 6.536993e-13
       768   0.088799  10.202420  81.725634 7.673862e-13
       832   0.108039  10.661504  85.394544 8.242296e-13
       896   0.148664   9.677143  77.503547 9.379164e-13
       960   0.152327  11.616290  93.027125 9.947598e-13
```

We can see that parallelization has substantially increased both the flops and the bandwidth.

3. **Finding OpenMP bugs.** The homework repository contains five OpenMP problems that contain bugs. These files are in C, but they can be compiled with the C++ compiler. Try to find these bugs and fix them. Add a short comment to the code describing what was wrong and how you fixed the problem. Add the solutions to your repository using the naming convention `omp_solved`{2,...}`.c`, and provide a Makefile to compile the fixed example problems.
Solutions:
Fixed the bugs and added the comments describing the issue.

4. **OpenMP version of 2D Jacobi/Gauss-Seidel smoothing.** Implement first a serial and then an OpenMP version of the two-dimensional Jacobi and Gauss-Seidel smoothers. This is similar to the problem on the first homework assignment, but for the unit square domain $\Omega = (0,1)\times(0,1)$. For a given function $f:\Omega\to\mathbb{R}$, we aim to find $u:\Omega\to\mathbb{R}$ such that

$$-\Delta u := -(u_{xx}+u_{yy}) = f \text{ in } \Omega, \tag{1}$$

and $u(x,y)=0$ for all boundary points $(x,y)\in\partial\Omega := \{(x,y) : x=0 \text{ or } y=0 \text{ or } x=1 \text{ or } y=1\}$. We go through analogous arguments as in homework 1, where we used finite differences to discretize the one-dimensional version of (**??**). In two dimensions, we choose the uniformly spaced points $\{(x_i,y_j)=(ih,jh) : i,j=0,1,\dots,N,N+1\}\subset[0,1]\times[0,1]$, with $h=1/(N+1)$, and approximate $u(x_i,y_j)\approx u_{i,j}$ and $f(x_i,y_j)\approx f_{i,j}$, for $i,j=0,\dots,N+1$; see Figure **??** (left). Using Taylor expansions as in the one-dimensional case results in

$$-\Delta u(x_i,y_j) = \frac{-u(x_i-h,y_j)-u(x_i,y_j-h)+4u(x_i,y_j)-u(x_i+h,y_j)-u(x_i,y_j+h)}{h^2} + \text{h.o.t.},$$

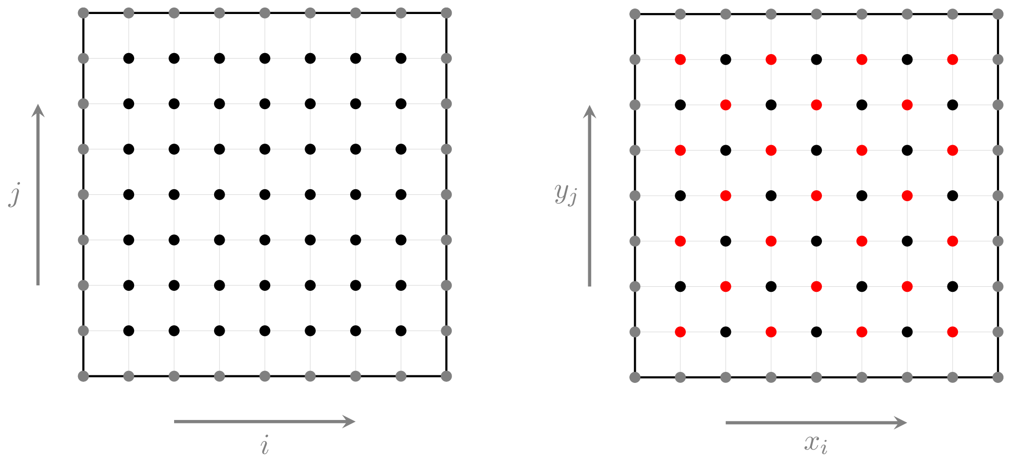


Figure 1: Sketch of discretization points for unit square for $N = 7$. Left: Dark points are unknowns, grey points at the boundary are zero. Right: red-black coloring of unknowns. Black and red points can be updated independently in a Gauss-Seidel step.

where h.o.t. stands for a remainder term that is of higher order in h, i.e., becomes small as h is decreased. Hence, we approximate the Laplace operator at a point (x_i, y_j) as follows:

$$-\Delta u_{ij} = \frac{-u_{i-1,j} - u_{i,j-1} + 4u_{ij} - u_{i+1,j} - u_{i,j+1}}{h^2}.$$

This results in a linear system, that can again be written as $A\boldsymbol{u} = \boldsymbol{f}$, where

$$\begin{aligned}\boldsymbol{u} &= (u_{1,1}, u_{1,2}, \dots, u_{1,N}, u_{2,1}, u_{2,2}, \dots, u_{N,N-1}, u_{N,N})^\top,\\ \boldsymbol{f} &= (f_{1,1}, f_{1,2}, \dots, f_{1,N}, f_{2,1}, f_{2,2}, \dots, f_{N,N-1}, f_{N,N})^\top.\end{aligned}$$

Note that the points at the boundaries are not included, as we know that their values to be zero. Similarly to the one-dimensional case, the resulting Jacobi update for solving this linear system is

$$u_{i,j}^{k+1} = \frac{1}{4}\left(h^2 f_{i,j} + u_{i-1,j}^k + u_{i,j-1}^k + u_{i+1,j}^k + u_{i,j+1}^k\right),$$

and the Gauss-Seidel update is given by

$$u_{i,j}^{k+1} = \frac{1}{4}\left(h^2 f_{i,j} + u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i+1,j}^k + u_{i,j+1}^k\right),$$

where it depens on the order of the unknowns which entries on the right hand side are based on the kth and which on the $(k+1)$st iteration. The above update formula is for lexicographic ordering of the points, i.e., we sweep from left to right first and go row by row from the bottom to the top. Usually, as in the one-dimensional case, one use a single vector $\boldsymbol{u}$ of unknows, which are overwritten and the latest available values are used.

As can be seen, the update at the (i, j)th point in the Gauss-Seidel smoother depends on previously updated points. This dependence makes it difficult to parallelize the Gauss-Seidel algorithm. As a remedy, we consider a variant of Gauss-Seidel, which uses *red-black* *coloring* of the unknowns. This amounts to "coloring" unknowns as shown in Figure **??** (right), and into splitting each Gauss-Seidel iteration into two sweeps: first, one updates all black and then all the red points (using the already updated red points). The point updates in the red and black sweeps are independent from each other and can be parallelized using OpenMP.[4] To detail the equations, this become the following update, where colors of the unknowns correspond to the colors of points in the figure, i.e., first we update all red points, i.e., (i, j) corresponds to indices for red points,

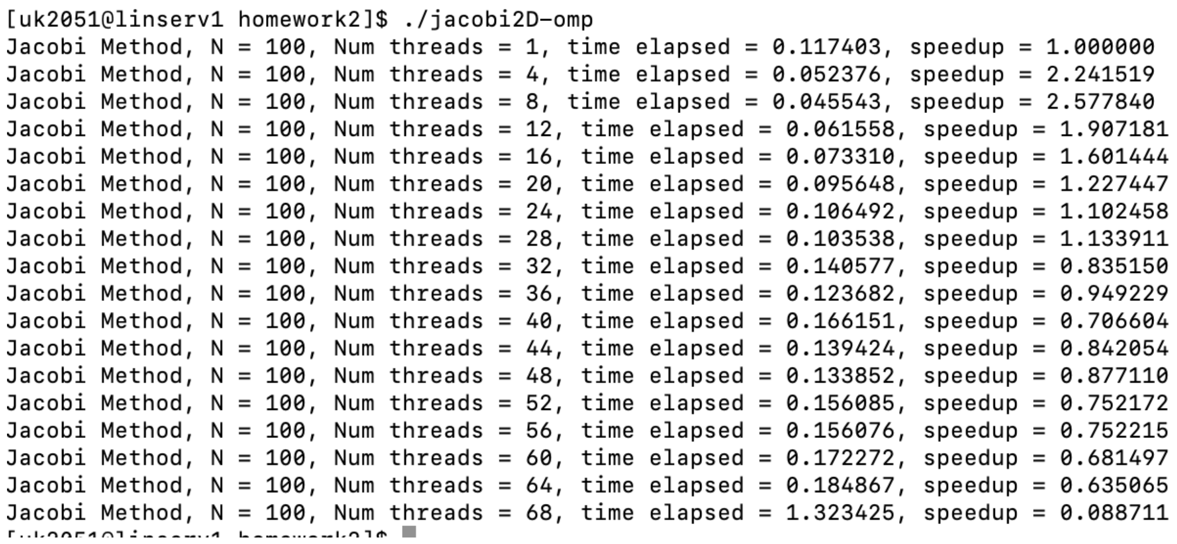
$$u_{i,j}^{k+1} = \frac{1}{4}\left(h^2 f_{i,j} + u_{i-1,j}^k + u_{i,j-1}^k + u_{i+1,j}^k + u_{i,j+1}^k\right),$$

and then we update all black points, i.e., (i, j) are indices corresponding to black points:

$$u_{i,j}^{k+1} = \frac{1}{4}\left(h^2 f_{i,j} + u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i+1,j}^{k+1} + u_{i,j+1}^{k+1}\right).$$

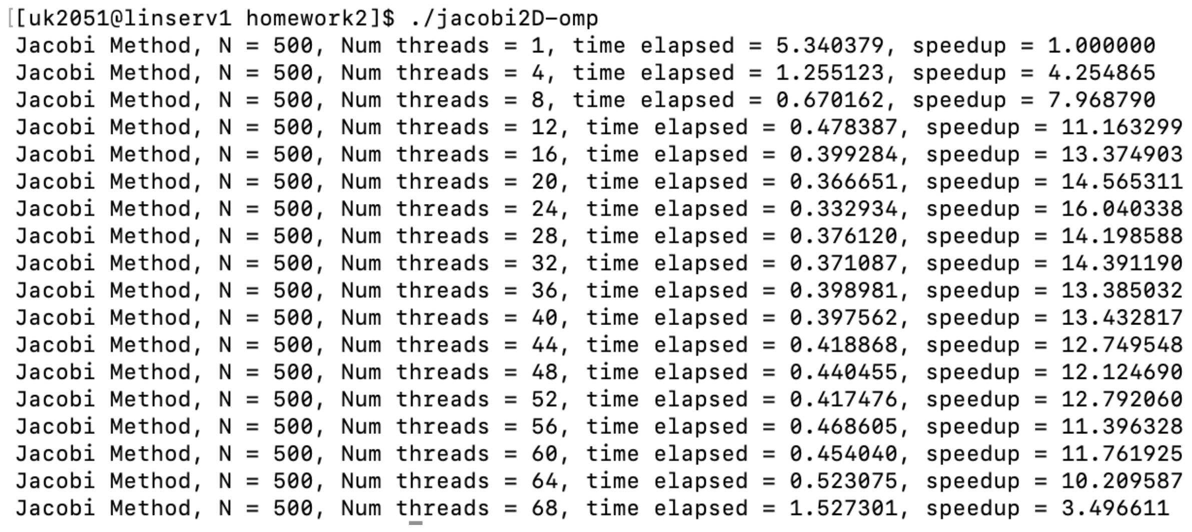
At the end, every point is on level $(n + 1)$ and we repeat.

- Write OpenMP implementations of the Jacobi and the Gauss-Seidel method with red-black coloring, and call them `jacobi2D-omp.cpp` and `gs2D-omp.cpp`. Make sure your OpenMP codes also compile without OpenMP compilers using preprocessor commands (`#ifdef _OPENMP`) as shown in class.
- Choose the right hand side $f(x, y) \equiv 1$, and report timings for different values of N and different numbers of threads, specifying the machine you run on. These timings should be for a fixed number of iterations as, similar to the 1D case, the convergence is slow, and slows down even further as N becomes larger.
  Solutions:
  Total number of iterations performed for both the Jacobi and Gauss Seidel are: 2000
  First is the data for Jacobi:

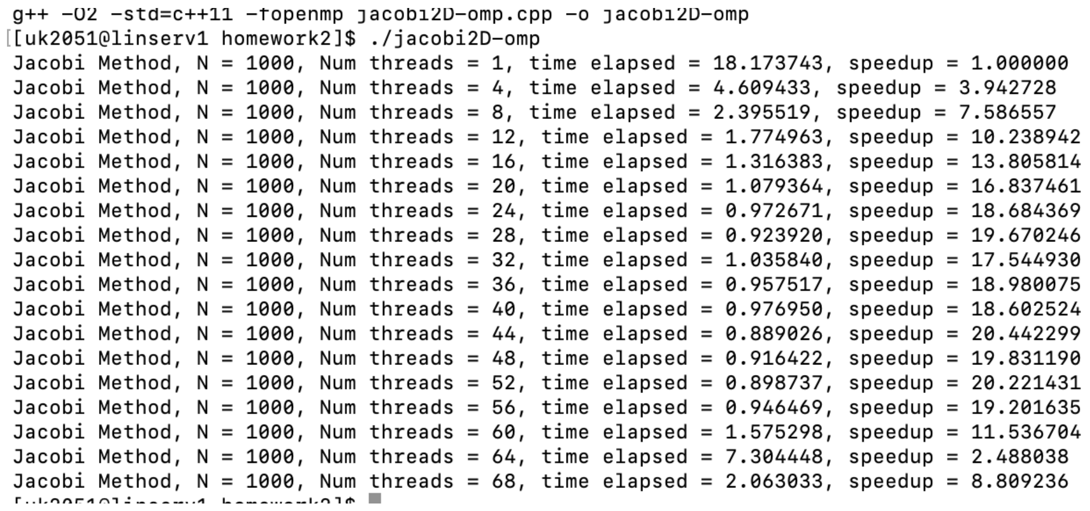


```
[uk2051@linserv1 homework2]$ ./jacobi2D-omp
Jacobi Method, N = 100, Num threads = 1, time elapsed = 0.117403, speedup = 1.000000
Jacobi Method, N = 100, Num threads = 4, time elapsed = 0.052376, speedup = 2.241519
Jacobi Method, N = 100, Num threads = 8, time elapsed = 0.045543, speedup = 2.577840
Jacobi Method, N = 100, Num threads = 12, time elapsed = 0.061558, speedup = 1.907181
Jacobi Method, N = 100, Num threads = 16, time elapsed = 0.073310, speedup = 1.601444
Jacobi Method, N = 100, Num threads = 20, time elapsed = 0.095648, speedup = 1.227447
Jacobi Method, N = 100, Num threads = 24, time elapsed = 0.106492, speedup = 1.102458
Jacobi Method, N = 100, Num threads = 28, time elapsed = 0.103538, speedup = 1.133911
Jacobi Method, N = 100, Num threads = 32, time elapsed = 0.140577, speedup = 0.835150
Jacobi Method, N = 100, Num threads = 36, time elapsed = 0.123682, speedup = 0.949229
Jacobi Method, N = 100, Num threads = 40, time elapsed = 0.166151, speedup = 0.706604
Jacobi Method, N = 100, Num threads = 44, time elapsed = 0.139424, speedup = 0.842054
Jacobi Method, N = 100, Num threads = 48, time elapsed = 0.133852, speedup = 0.877110
Jacobi Method, N = 100, Num threads = 52, time elapsed = 0.156085, speedup = 0.752172
Jacobi Method, N = 100, Num threads = 56, time elapsed = 0.156076, speedup = 0.752215
Jacobi Method, N = 100, Num threads = 60, time elapsed = 0.172272, speedup = 0.681497
Jacobi Method, N = 100, Num threads = 64, time elapsed = 0.184867, speedup = 0.635065
Jacobi Method, N = 100, Num threads = 68, time elapsed = 1.323425, speedup = 0.088711
[uk2051@linserv1 homework2]$
```

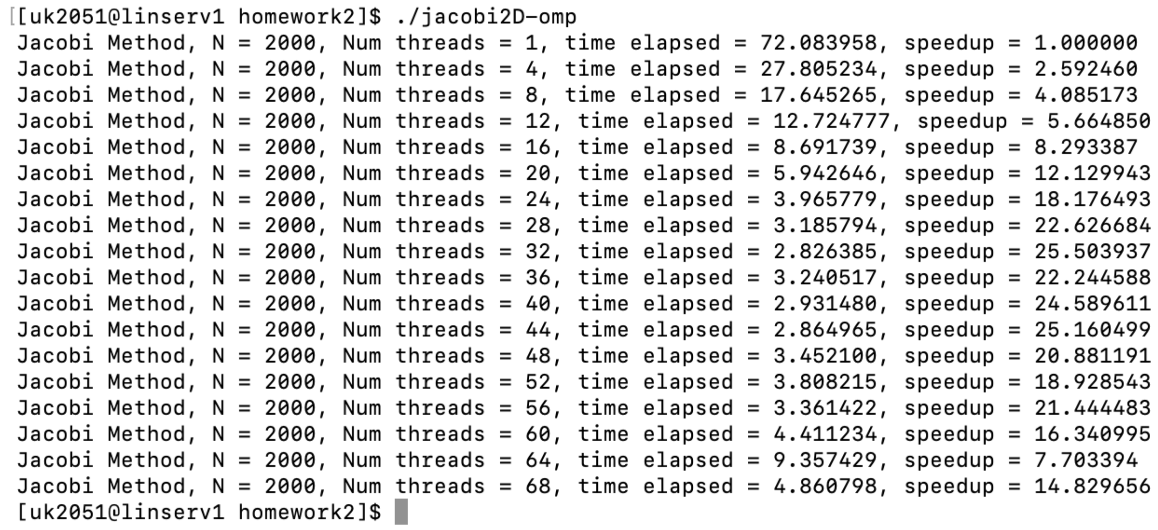
[4]Depending on the discretization and the dimension of the problem, one might require more than two colors to ensure that updates become independent from each other and allow for parallelism. Efficient coloring for unstructured meshes with as little colors as possible is a difficult research question.



```
[[uk2051@linserv1 homework2]$ ./jacobi2D-omp
 Jacobi Method, N = 500, Num threads = 1, time elapsed = 5.340379, speedup = 1.000000
 Jacobi Method, N = 500, Num threads = 4, time elapsed = 1.255123, speedup = 4.254865
 Jacobi Method, N = 500, Num threads = 8, time elapsed = 0.670162, speedup = 7.968790
 Jacobi Method, N = 500, Num threads = 12, time elapsed = 0.478387, speedup = 11.163299
 Jacobi Method, N = 500, Num threads = 16, time elapsed = 0.399284, speedup = 13.374903
 Jacobi Method, N = 500, Num threads = 20, time elapsed = 0.366651, speedup = 14.565311
 Jacobi Method, N = 500, Num threads = 24, time elapsed = 0.332934, speedup = 16.040338
 Jacobi Method, N = 500, Num threads = 28, time elapsed = 0.376120, speedup = 14.198588
 Jacobi Method, N = 500, Num threads = 32, time elapsed = 0.371087, speedup = 14.391190
 Jacobi Method, N = 500, Num threads = 36, time elapsed = 0.398981, speedup = 13.385032
 Jacobi Method, N = 500, Num threads = 40, time elapsed = 0.397562, speedup = 13.432817
 Jacobi Method, N = 500, Num threads = 44, time elapsed = 0.418868, speedup = 12.749548
 Jacobi Method, N = 500, Num threads = 48, time elapsed = 0.440455, speedup = 12.124690
 Jacobi Method, N = 500, Num threads = 52, time elapsed = 0.417476, speedup = 12.792060
 Jacobi Method, N = 500, Num threads = 56, time elapsed = 0.468605, speedup = 11.396328
 Jacobi Method, N = 500, Num threads = 60, time elapsed = 0.454040, speedup = 11.761925
 Jacobi Method, N = 500, Num threads = 64, time elapsed = 0.523075, speedup = 10.209587
 Jacobi Method, N = 500, Num threads = 68, time elapsed = 1.527301, speedup = 3.496611
```

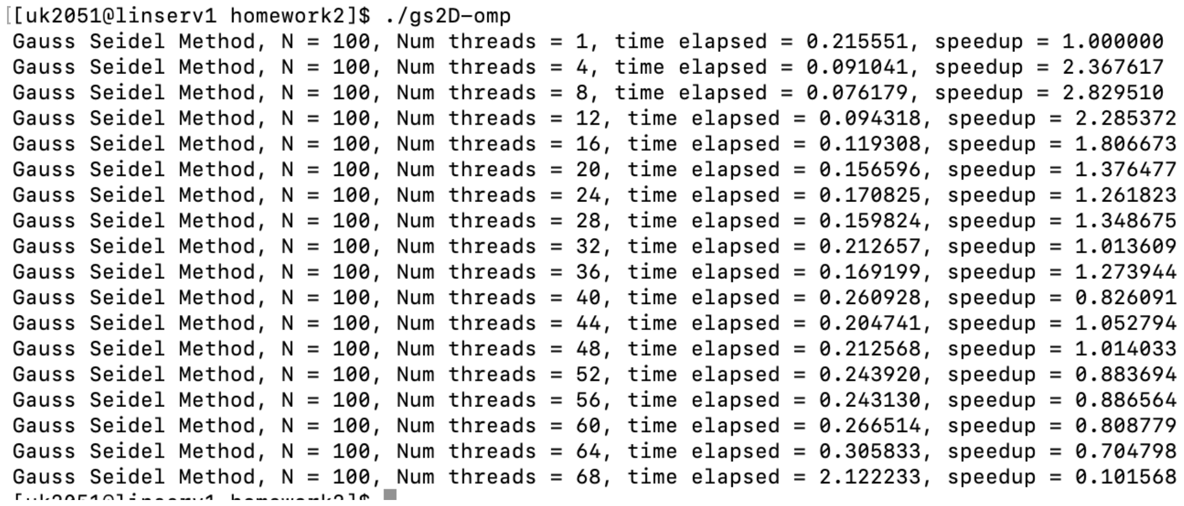


```
 g++ -O2 -std=c++11 -fopenmp jacobi2D-omp.cpp -o jacobi2D-omp
[[uk2051@linserv1 homework2]$ ./jacobi2D-omp
 Jacobi Method, N = 1000, Num threads = 1, time elapsed = 18.173743, speedup = 1.000000
 Jacobi Method, N = 1000, Num threads = 4, time elapsed = 4.609433, speedup = 3.942728
 Jacobi Method, N = 1000, Num threads = 8, time elapsed = 2.395519, speedup = 7.586557
 Jacobi Method, N = 1000, Num threads = 12, time elapsed = 1.774963, speedup = 10.238942
 Jacobi Method, N = 1000, Num threads = 16, time elapsed = 1.316383, speedup = 13.805814
 Jacobi Method, N = 1000, Num threads = 20, time elapsed = 1.079364, speedup = 16.837461
 Jacobi Method, N = 1000, Num threads = 24, time elapsed = 0.972671, speedup = 18.684369
 Jacobi Method, N = 1000, Num threads = 28, time elapsed = 0.923920, speedup = 19.670246
 Jacobi Method, N = 1000, Num threads = 32, time elapsed = 1.035840, speedup = 17.544930
 Jacobi Method, N = 1000, Num threads = 36, time elapsed = 0.957517, speedup = 18.980075
 Jacobi Method, N = 1000, Num threads = 40, time elapsed = 0.976950, speedup = 18.602524
 Jacobi Method, N = 1000, Num threads = 44, time elapsed = 0.889026, speedup = 20.442299
 Jacobi Method, N = 1000, Num threads = 48, time elapsed = 0.916422, speedup = 19.831190
 Jacobi Method, N = 1000, Num threads = 52, time elapsed = 0.898737, speedup = 20.221431
 Jacobi Method, N = 1000, Num threads = 56, time elapsed = 0.946469, speedup = 19.201635
 Jacobi Method, N = 1000, Num threads = 60, time elapsed = 1.575298, speedup = 11.536704
 Jacobi Method, N = 1000, Num threads = 64, time elapsed = 7.304448, speedup = 2.488038
 Jacobi Method, N = 1000, Num threads = 68, time elapsed = 2.063033, speedup = 8.809236
```

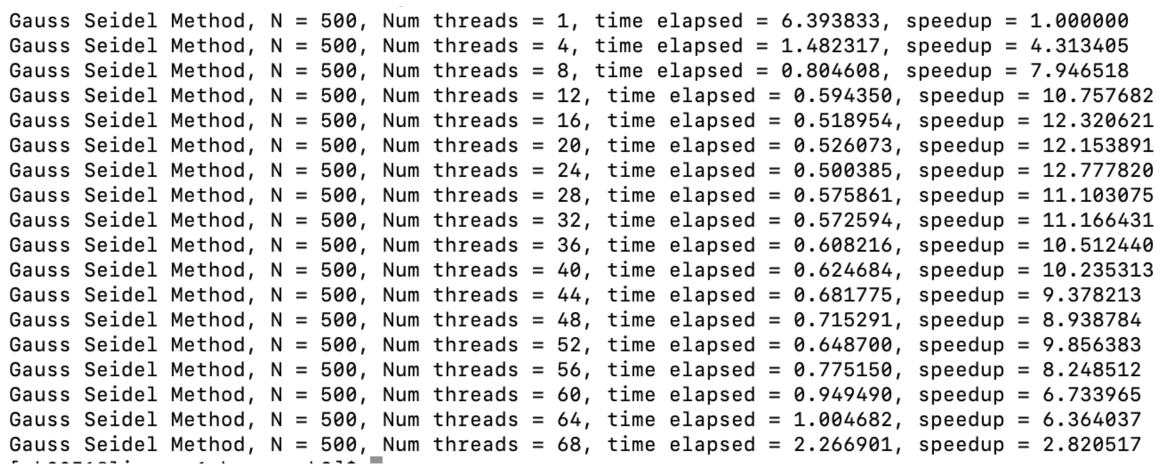


```
[[uk2051@linserv1 homework2]$ ./jacobi2D-omp
 Jacobi Method, N = 2000, Num threads = 1, time elapsed = 72.083958, speedup = 1.000000
 Jacobi Method, N = 2000, Num threads = 4, time elapsed = 27.805234, speedup = 2.592460
 Jacobi Method, N = 2000, Num threads = 8, time elapsed = 17.645265, speedup = 4.085173
 Jacobi Method, N = 2000, Num threads = 12, time elapsed = 12.724777, speedup = 5.664850
 Jacobi Method, N = 2000, Num threads = 16, time elapsed = 8.691739, speedup = 8.293387
 Jacobi Method, N = 2000, Num threads = 20, time elapsed = 5.942646, speedup = 12.129943
 Jacobi Method, N = 2000, Num threads = 24, time elapsed = 3.965779, speedup = 18.176493
 Jacobi Method, N = 2000, Num threads = 28, time elapsed = 3.185794, speedup = 22.626684
 Jacobi Method, N = 2000, Num threads = 32, time elapsed = 2.826385, speedup = 25.503937
 Jacobi Method, N = 2000, Num threads = 36, time elapsed = 3.240517, speedup = 22.244588
 Jacobi Method, N = 2000, Num threads = 40, time elapsed = 2.931480, speedup = 24.589611
 Jacobi Method, N = 2000, Num threads = 44, time elapsed = 2.864965, speedup = 25.160499
 Jacobi Method, N = 2000, Num threads = 48, time elapsed = 3.452100, speedup = 20.881191
 Jacobi Method, N = 2000, Num threads = 52, time elapsed = 3.808215, speedup = 18.928543
 Jacobi Method, N = 2000, Num threads = 56, time elapsed = 3.361422, speedup = 21.444483
 Jacobi Method, N = 2000, Num threads = 60, time elapsed = 4.411234, speedup = 16.340995
 Jacobi Method, N = 2000, Num threads = 64, time elapsed = 9.357429, speedup = 7.703394
 Jacobi Method, N = 2000, Num threads = 68, time elapsed = 4.860798, speedup = 14.829656
 [uk2051@linserv1 homework2]$
```

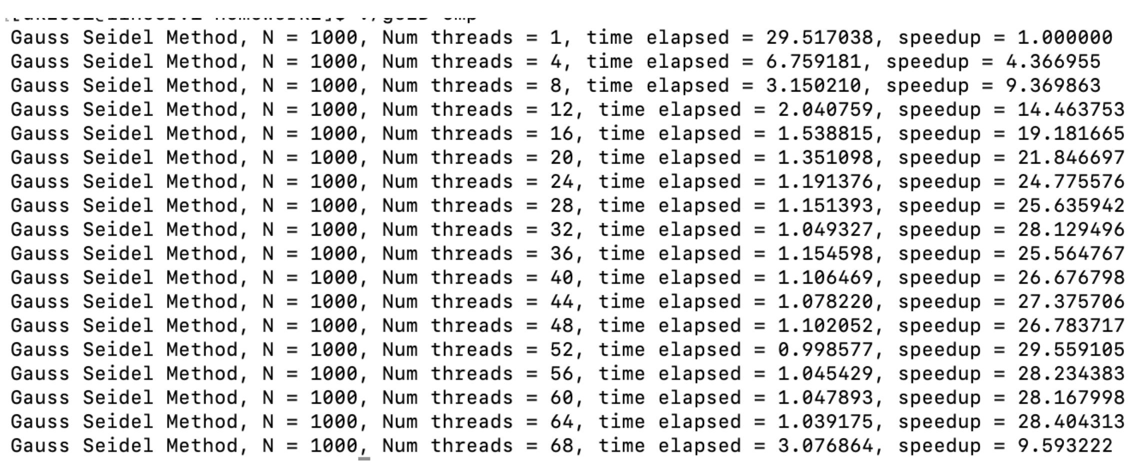
Below is the data for Gauss Seidel:



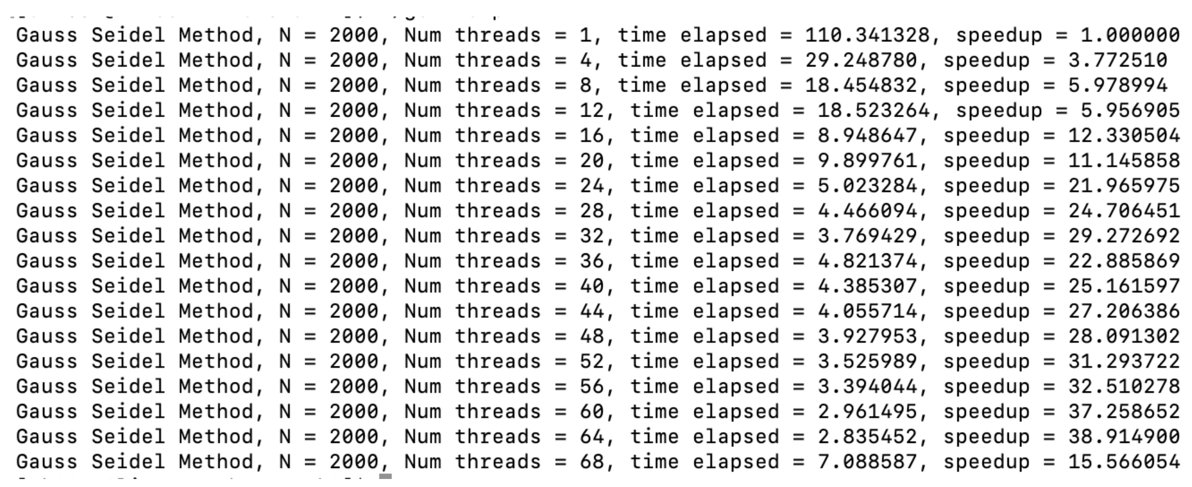
```
[[uk2051@linserv1 homework2]$ ./gs2D-omp
Gauss Seidel Method, N = 100, Num threads = 1, time elapsed = 0.215551, speedup = 1.000000
Gauss Seidel Method, N = 100, Num threads = 4, time elapsed = 0.091041, speedup = 2.367617
Gauss Seidel Method, N = 100, Num threads = 8, time elapsed = 0.076179, speedup = 2.829510
Gauss Seidel Method, N = 100, Num threads = 12, time elapsed = 0.094318, speedup = 2.285372
Gauss Seidel Method, N = 100, Num threads = 16, time elapsed = 0.119308, speedup = 1.806673
Gauss Seidel Method, N = 100, Num threads = 20, time elapsed = 0.156596, speedup = 1.376477
Gauss Seidel Method, N = 100, Num threads = 24, time elapsed = 0.170825, speedup = 1.261823
Gauss Seidel Method, N = 100, Num threads = 28, time elapsed = 0.159824, speedup = 1.348675
Gauss Seidel Method, N = 100, Num threads = 32, time elapsed = 0.212657, speedup = 1.013609
Gauss Seidel Method, N = 100, Num threads = 36, time elapsed = 0.169199, speedup = 1.273944
Gauss Seidel Method, N = 100, Num threads = 40, time elapsed = 0.260928, speedup = 0.826091
Gauss Seidel Method, N = 100, Num threads = 44, time elapsed = 0.204741, speedup = 1.052794
Gauss Seidel Method, N = 100, Num threads = 48, time elapsed = 0.212568, speedup = 1.014033
Gauss Seidel Method, N = 100, Num threads = 52, time elapsed = 0.243920, speedup = 0.883694
Gauss Seidel Method, N = 100, Num threads = 56, time elapsed = 0.243130, speedup = 0.886564
Gauss Seidel Method, N = 100, Num threads = 60, time elapsed = 0.266514, speedup = 0.808779
Gauss Seidel Method, N = 100, Num threads = 64, time elapsed = 0.305833, speedup = 0.704798
Gauss Seidel Method, N = 100, Num threads = 68, time elapsed = 2.122233, speedup = 0.101568
```



```
Gauss Seidel Method, N = 500, Num threads = 1, time elapsed = 6.393833, speedup = 1.000000
Gauss Seidel Method, N = 500, Num threads = 4, time elapsed = 1.482317, speedup = 4.313405
Gauss Seidel Method, N = 500, Num threads = 8, time elapsed = 0.804608, speedup = 7.946518
Gauss Seidel Method, N = 500, Num threads = 12, time elapsed = 0.594350, speedup = 10.757682
Gauss Seidel Method, N = 500, Num threads = 16, time elapsed = 0.518954, speedup = 12.320621
Gauss Seidel Method, N = 500, Num threads = 20, time elapsed = 0.526073, speedup = 12.153891
Gauss Seidel Method, N = 500, Num threads = 24, time elapsed = 0.500385, speedup = 12.777820
Gauss Seidel Method, N = 500, Num threads = 28, time elapsed = 0.575861, speedup = 11.103075
Gauss Seidel Method, N = 500, Num threads = 32, time elapsed = 0.572594, speedup = 11.166431
Gauss Seidel Method, N = 500, Num threads = 36, time elapsed = 0.608216, speedup = 10.512440
Gauss Seidel Method, N = 500, Num threads = 40, time elapsed = 0.624684, speedup = 10.235313
Gauss Seidel Method, N = 500, Num threads = 44, time elapsed = 0.681775, speedup = 9.378213
Gauss Seidel Method, N = 500, Num threads = 48, time elapsed = 0.715291, speedup = 8.938784
Gauss Seidel Method, N = 500, Num threads = 52, time elapsed = 0.648700, speedup = 9.856383
Gauss Seidel Method, N = 500, Num threads = 56, time elapsed = 0.775150, speedup = 8.248512
Gauss Seidel Method, N = 500, Num threads = 60, time elapsed = 0.949490, speedup = 6.733965
Gauss Seidel Method, N = 500, Num threads = 64, time elapsed = 1.004682, speedup = 6.364037
Gauss Seidel Method, N = 500, Num threads = 68, time elapsed = 2.266901, speedup = 2.820517
```



```
Gauss Seidel Method, N = 1000, Num threads = 1, time elapsed = 29.517038, speedup = 1.000000
Gauss Seidel Method, N = 1000, Num threads = 4, time elapsed = 6.759181, speedup = 4.366955
Gauss Seidel Method, N = 1000, Num threads = 8, time elapsed = 3.150210, speedup = 9.369863
Gauss Seidel Method, N = 1000, Num threads = 12, time elapsed = 2.040759, speedup = 14.463753
Gauss Seidel Method, N = 1000, Num threads = 16, time elapsed = 1.538815, speedup = 19.181665
Gauss Seidel Method, N = 1000, Num threads = 20, time elapsed = 1.351098, speedup = 21.846697
Gauss Seidel Method, N = 1000, Num threads = 24, time elapsed = 1.191376, speedup = 24.775576
Gauss Seidel Method, N = 1000, Num threads = 28, time elapsed = 1.151393, speedup = 25.635942
Gauss Seidel Method, N = 1000, Num threads = 32, time elapsed = 1.049327, speedup = 28.129496
Gauss Seidel Method, N = 1000, Num threads = 36, time elapsed = 1.154598, speedup = 25.564767
Gauss Seidel Method, N = 1000, Num threads = 40, time elapsed = 1.106469, speedup = 26.676798
Gauss Seidel Method, N = 1000, Num threads = 44, time elapsed = 1.078220, speedup = 27.375706
Gauss Seidel Method, N = 1000, Num threads = 48, time elapsed = 1.102052, speedup = 26.783717
Gauss Seidel Method, N = 1000, Num threads = 52, time elapsed = 0.998577, speedup = 29.559105
Gauss Seidel Method, N = 1000, Num threads = 56, time elapsed = 1.045429, speedup = 28.234383
Gauss Seidel Method, N = 1000, Num threads = 60, time elapsed = 1.047893, speedup = 28.167998
Gauss Seidel Method, N = 1000, Num threads = 64, time elapsed = 1.039175, speedup = 28.404313
Gauss Seidel Method, N = 1000, Num threads = 68, time elapsed = 3.076864, speedup = 9.593222
```



```
Gauss Seidel Method, N = 2000, Num threads = 1, time elapsed = 110.341328, speedup = 1.000000
Gauss Seidel Method, N = 2000, Num threads = 4, time elapsed = 29.248780, speedup = 3.772510
Gauss Seidel Method, N = 2000, Num threads = 8, time elapsed = 18.454832, speedup = 5.978994
Gauss Seidel Method, N = 2000, Num threads = 12, time elapsed = 18.523264, speedup = 5.956905
Gauss Seidel Method, N = 2000, Num threads = 16, time elapsed = 8.948647, speedup = 12.330504
Gauss Seidel Method, N = 2000, Num threads = 20, time elapsed = 9.899761, speedup = 11.145858
Gauss Seidel Method, N = 2000, Num threads = 24, time elapsed = 5.023284, speedup = 21.965975
Gauss Seidel Method, N = 2000, Num threads = 28, time elapsed = 4.466094, speedup = 24.706451
Gauss Seidel Method, N = 2000, Num threads = 32, time elapsed = 3.769429, speedup = 29.272692
Gauss Seidel Method, N = 2000, Num threads = 36, time elapsed = 4.821374, speedup = 22.885869
Gauss Seidel Method, N = 2000, Num threads = 40, time elapsed = 4.385307, speedup = 25.161597
Gauss Seidel Method, N = 2000, Num threads = 44, time elapsed = 4.055714, speedup = 27.206386
Gauss Seidel Method, N = 2000, Num threads = 48, time elapsed = 3.927953, speedup = 28.091302
Gauss Seidel Method, N = 2000, Num threads = 52, time elapsed = 3.525989, speedup = 31.293722
Gauss Seidel Method, N = 2000, Num threads = 56, time elapsed = 3.394044, speedup = 32.510278
Gauss Seidel Method, N = 2000, Num threads = 60, time elapsed = 2.961495, speedup = 37.258652
Gauss Seidel Method, N = 2000, Num threads = 64, time elapsed = 2.835452, speedup = 38.914900
Gauss Seidel Method, N = 2000, Num threads = 68, time elapsed = 7.088587, speedup = 15.566054
```