CFD Project 01 Update

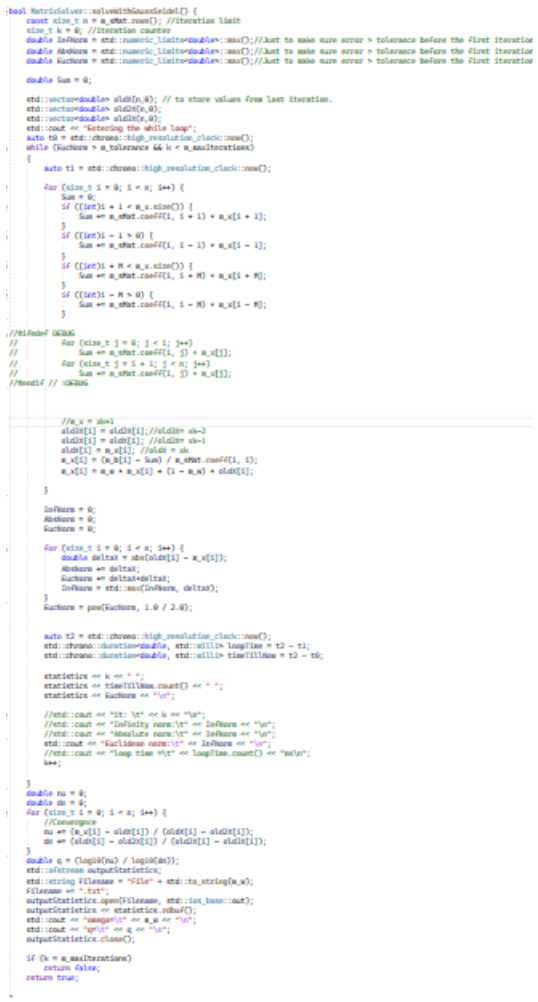
By: Arshia Saffari

This update is actually already used in the original project to solve the 400 by 400 grid. Gauss-Seidel method is now modified to only multiply the potentially non-zero elements of the sparce matrix to their respective unknown values. The performance gain is huge.

```
void HeatEquationSolver::createEquationMatrix() {
    // 5 elements per row
    for (size_t i = 0; i < m_eqMatDims; i++)
        m_eqKnownVec.push_back(0);
    std::vector<Eigen::Triplet<double> > Vec;
    Vec.reserve(m_eqMatDims * 5);
    size_t N = m_nodeCount.ycount - 2;
    size_t M = m_nodeCount.xcount - 2;
    double L = m_computeDomain.xrange.second - m_computeDomain.xrange.first;
    double H = m_computeDomain.yrange.second - m_computeDomain.yrange.first;
    double gamma = L * (N + 1) / (H * (M + 1));
    //for (size_t i = 0; i < M * N; i++)
    //  for (size_t j = 0; j < M * N; j++)
    //      //m_eqMat.A[i][j] = 0;
    for (size_t i = 0; i < M; i++) {
        for (size_t j = 0; j < N; j++) {
            // for node i,j:
            if (j == 0) {
                //m_eqMat.b[j * M + i] -= 0 * gamma;
                //m_eqMat.A[j * M + i][(j + 1) * M + i] = 1 * gamma;
                m_eqKnownVec[j * M + i] -= 0 * gamma;
                Vec.push_back(Eigen::Triplet<double>(j * M + i, (j + 1) * M + i, 1 * gamma));
            }
            else if (j == N - 1) {
                //m_eqMat.b[j * M + i] -= 1 * gamma;
                //m_eqMat.A[j * M + i][(j - 1) * M + i] = 1 * gamma;
                m_eqKnownVec[j * M + i] -= 1 * gamma;
                Vec.push_back(Eigen::Triplet<double>(j * M + i, (j - 1) * M + i, 1 * gamma));
            }
            else {
                //m_eqMat.A[j * M + i][(j - 1) * M + i] = 1 * gamma;
                //m_eqMat.A[j * M + i][(j + 1) * M + i] = 1 * gamma;
                Vec.push_back(Eigen::Triplet<double>(j * M + i, (j - 1) * M + i, 1 * gamma));
                Vec.push_back(Eigen::Triplet<double>(j * M + i, (j + 1) * M + i, 1 * gamma));

            }
            if (i == 0) {
                //m_eqMat.b[j * M + i] -= 0;
                //m_eqMat.A[j * M + i][j * M + i + 1] = 1;
                m_eqKnownVec[j * M + i] -= 0;
                Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i + 1, 1));
            }
            else if (i == M - 1) {
                //m_eqMat.b[j * M + i] -= 0;
                //m_eqMat.A[j * M + i][j * M + i - 1] = 1;
                m_eqKnownVec[j * M + i] -= 0;
                Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i - 1, 1));
            }
            else {
                //m_eqMat.A[j * M + i][j * M + i - 1] = 1;
                //m_eqMat.A[j * M + i][j * M + i + 1] = 1;
                Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i - 1, 1));
                Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i + 1, 1));
            }
            //m_eqMat.A[j * M + i][j * M + i] = -4;
            Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i, -4));
        }// for j
    }// for i
    m_eqSparseMat.resize(N * M, N * M);
    m_eqSparseMat.setFromTriplets(Vec.begin(), Vec.end());
```

Basically, looking at the matrix generator code there are only 5 elements per row at best. These are located at $(i,i),(i,i+1),(i,i-1),(i,i+M),(i,i-M)$. The parameter M is passed to solver and instead of multiplying each element (which for a 400 by 400 grid is $398 \times 398 = 158404$ elements in each of the 158404 rows of the matrix of which, 158403 elements are multiplied to their respective unknown vector value, summed and subtracted from respective known vector element and divided by $A(i,i)$) only 4 elements (depending on the boundary condition they might still be 0 but it's not worth the effort to identify them) are multiplied. This approach reduces the time considerably.

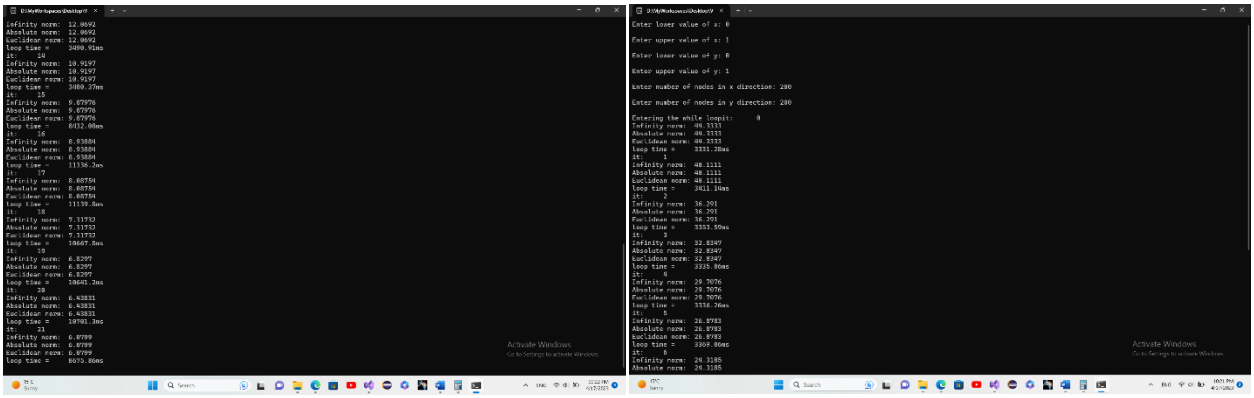


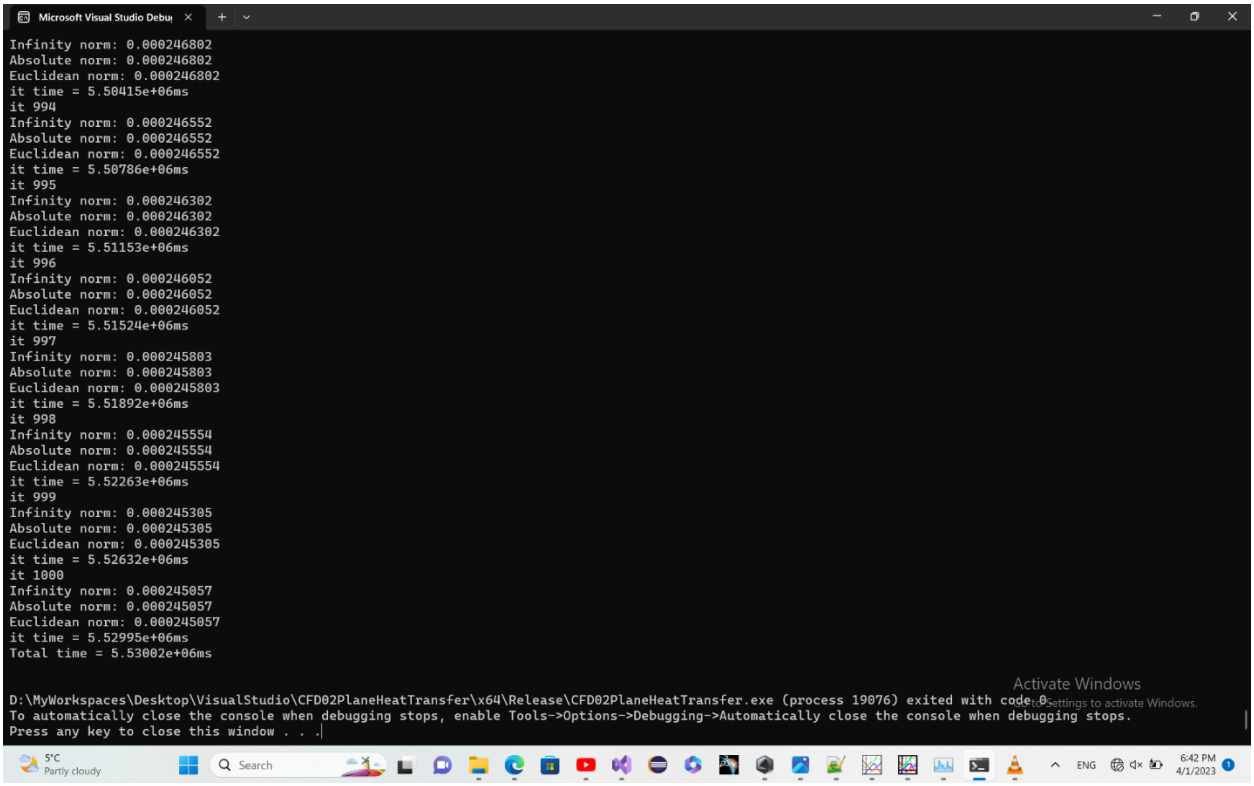
The new solver codes.

Performance comparison:

Old solver:

A simple 200 by 200 matrix takes a minimum of about 3331.28ms per iteration which grows up significantly at times even as much as 11139.8ms.





It took about 1.5 hours to complete with a accuracy of 1e-4.

New code:

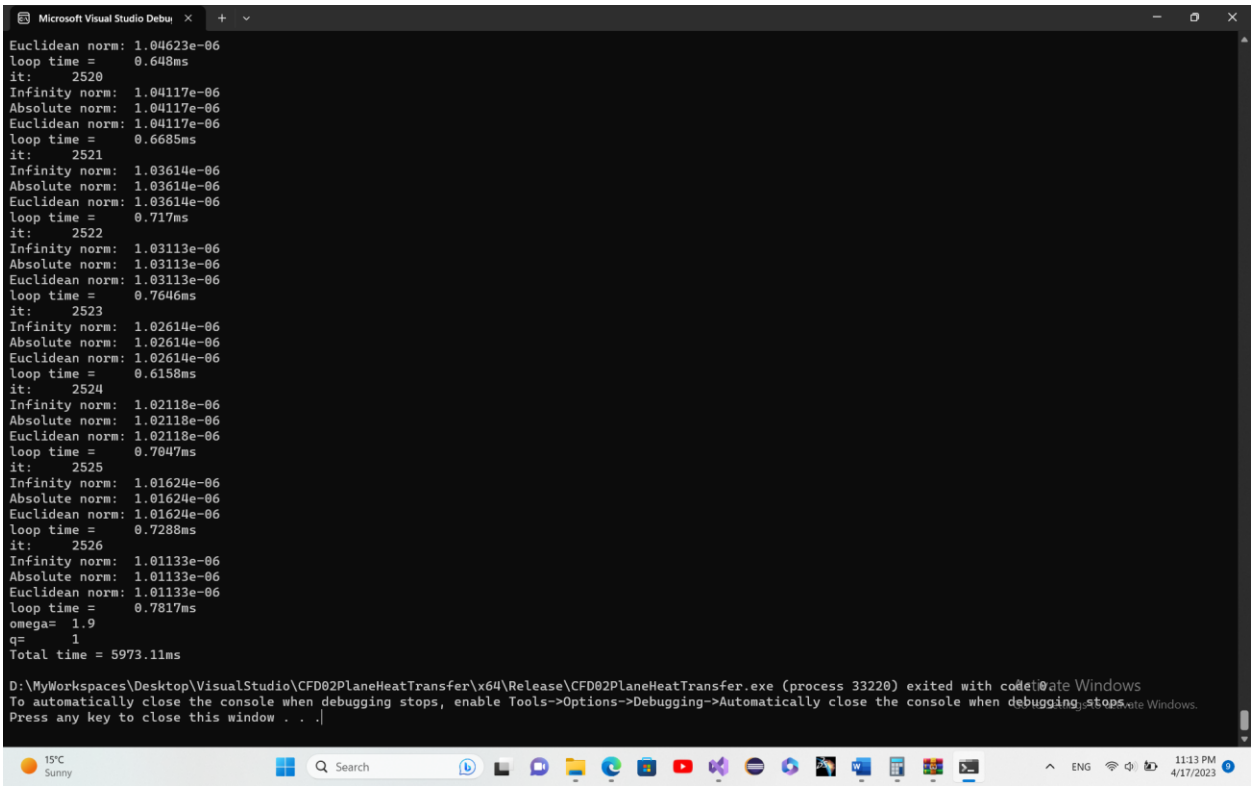
Replacing the old method:

```
//for (size_t j = 0; j < i; j++)
//  Sum += m_sMat.coeff(i, j) * m_x[j];
//for (size_t j = i + 1; j < n; j++)
//  Sum += m_sMat.coeff(i, j) * m_x[j];
```

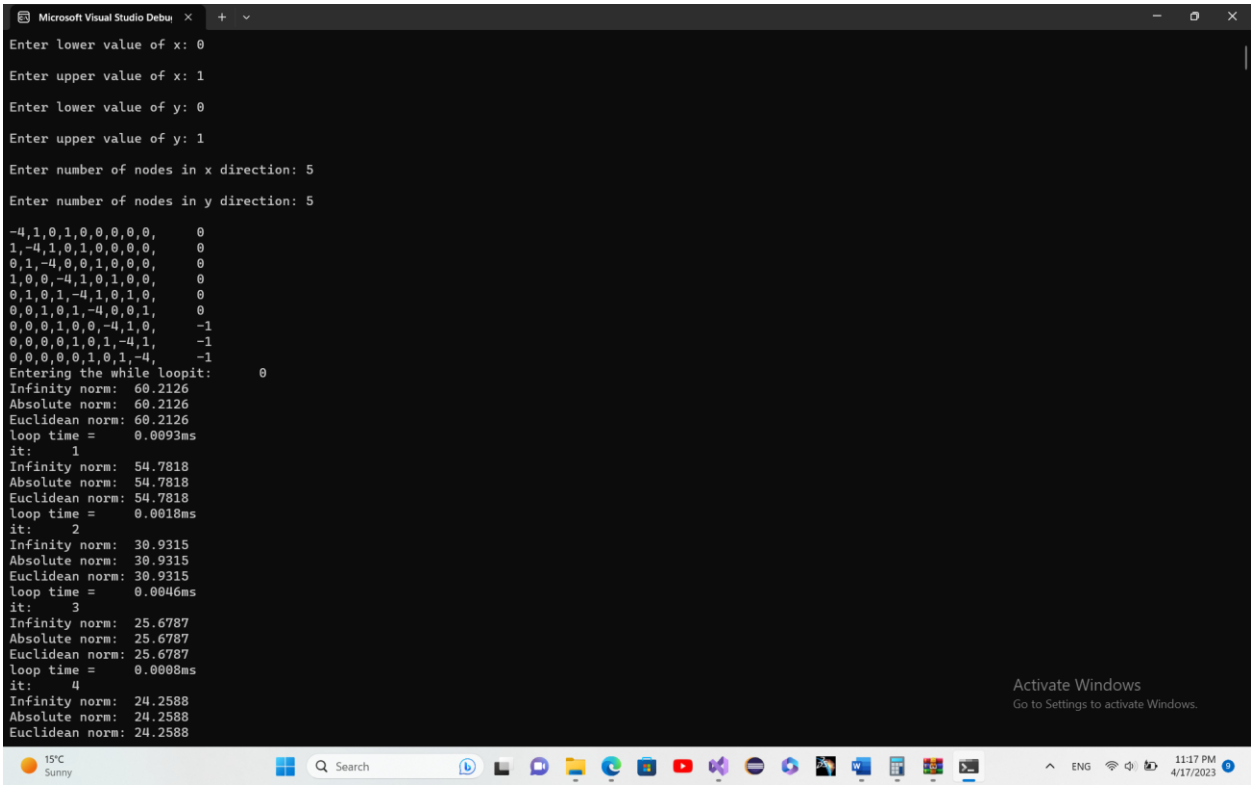
With newer one

```
if ((int)i + 1 < m_x.size()) {
    Sum += m_sMat.coeff(i, i + 1) * m_x[i + 1];
}
if ((int)i - 1 > 0) {
    Sum += m_sMat.coeff(i, i - 1) * m_x[i - 1];
}
if ((int)i + M < m_x.size()) {
    Sum += m_sMat.coeff(i, i + M) * m_x[i + M];
}
if ((int)i - M > 0) {
    Sum += m_sMat.coeff(i, i - M) * m_x[i - M];
}
```

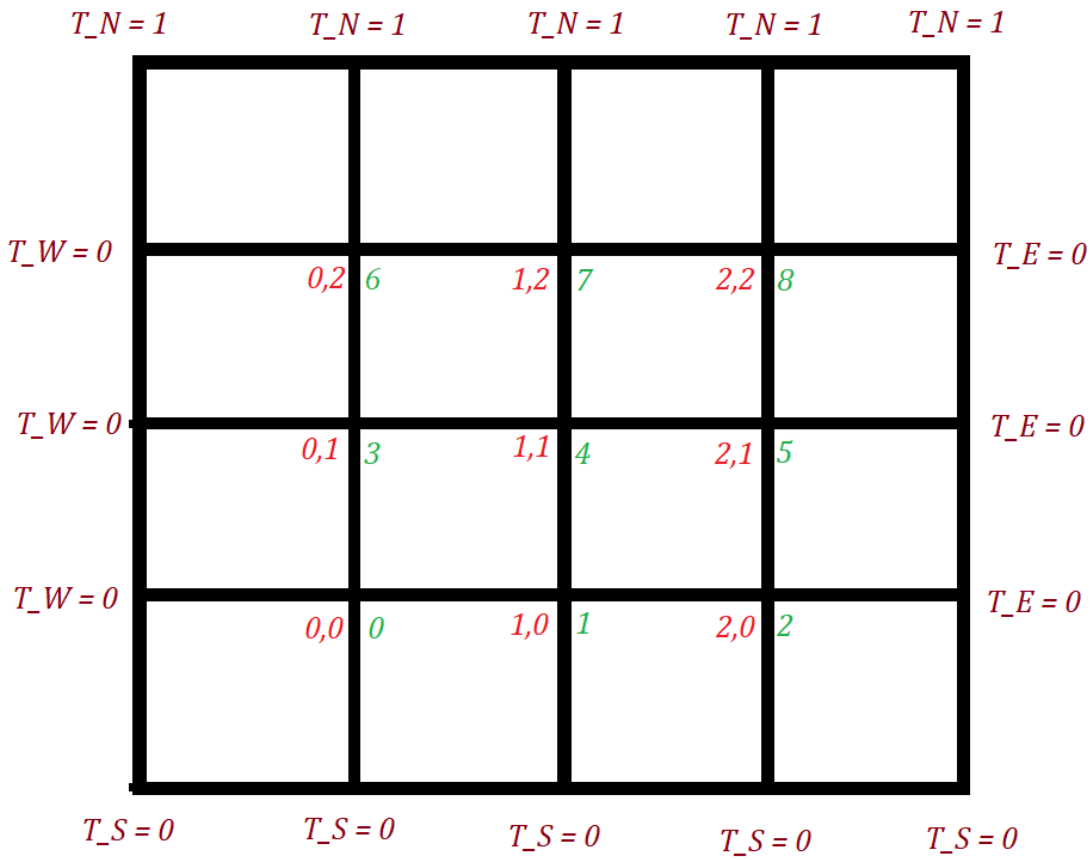
This loop takes about 0.5 to 0.7ms to complete on iteration and takes about 6 seconds to complete 2526 iterations which is much more than what was previously possible and is much more accurate with the error being less than 1e-6. The bottleneck at this point is consul print.



Some more talks about the matrix:



This is the sparse matrix generated by `void HeatEquationSolver::createEquationMatrix()` method for a 5 by 5 gird.



The indexing (2D index in red on bottom left of each node and 1D index on bottom right). Sorry for the bad paint job.

```
-4,1,0,1,0,0,0,0,0,      0
1,-4,1,0,1,0,0,0,0,      0
0,1,-4,0,0,1,0,0,0,      0
1,0,0,-4,1,0,1,0,0,      0
0,1,0,1,-4,1,0,1,0,      0
0,0,1,0,1,-4,0,0,1,      0
0,0,0,1,0,0,-4,1,0,      -1
0,0,0,0,1,0,1,-4,1,      -1
0,0,0,0,0,1,0,1,-4,      -1
```

As expected, the equations are

$$\text{Line 0: } -4T_0 + 1T_1 + 0T_2 + 1T_3 + 0T_4 + 0T_5 + 0T_6 + 0T_7 + 0T_8 = 0$$

$$\text{Line 1: } 1T_0 - 4T_1 + 1T_2 + 0T_3 + 1T_4 + 0T_5 + 0T_6 + 0T_7 + 0T_8 = 0$$

$$\text{Line 2: } 0T_0 + 1T_1 - 4T_2 + 0T_3 + 0T_4 + 1T_5 + 0T_6 + 0T_7 + 0T_8 = 0$$

$$\text{Line 3: } 1T_0 + 0T_1 + 0T_2 - 4T_3 + 1T_4 + 0T_5 + 1T_6 + 0T_7 + 0T_8 = 0$$

$$\text{Line 4: } 0T_0 + 1T_1 + 0T_2 + 1T_3 - 4T_4 + 1T_5 + 0T_6 + 1T_7 + 0T_8 = 0$$

$$\text{Line 5: } 0T_0 + 0T_1 + 1T_2 + 0T_3 + 1T_4 - 4T_5 + 0T_6 + 0T_7 + 1T_8 = 0$$

$$\text{Line 6: } 0T_0 + 0T_1 + 0T_2 + 1T_3 + 0T_4 + 0T_5 - 4T_6 + 1T_7 + 0T_8 = -1$$

$$\text{Line 7: } 0T_0 + 0T_1 + 0T_2 + 0T_3 + 1T_4 + 0T_5 + 1T_6 - 4T_7 + 1T_8 = -1$$

$$\text{Line 8: } 0T_0 + 0T_1 + 0T_2 + 0T_3 + 0T_4 + 1T_5 + 0T_6 + 1T_7 - 4T_8 = -1$$

Which seems true as equations 6,7 and 8 are the only ones with -1 in the known vector (from moving +1 temperature in the north node to the RHS.)

Each line is written for each node and all nodes are unknown temperature, the -4 coefficient starts form line 0 and T0 to line 8 and T8.

And the neighboring nodes have coefficients equal to 1.