Documentation for the main change done at revision 8d0738f6f00751044a1789b8fbdd12ecf918e8c8

I will denote the 3 matrices M, X and Y using only one picture, in which each cell will contain the 3 correspondent values of the matrices M, X and Y in that cell. The values of M, X and Y are going to be positiones as follows:

In the current computation, we initialize initialize the first row and first colum of the matrices M, X, and Y and then we proceed to compute the remaining cells of the matrix. The values for the first row and column initialization are:

0 0 k	0 0 k	0 0 k	0 0 k	0 0 k	0 0 k	0 0 k	0 0 k	0 0 k
0 0 0								
0 0 0								
0 0 0								
0 0 0								

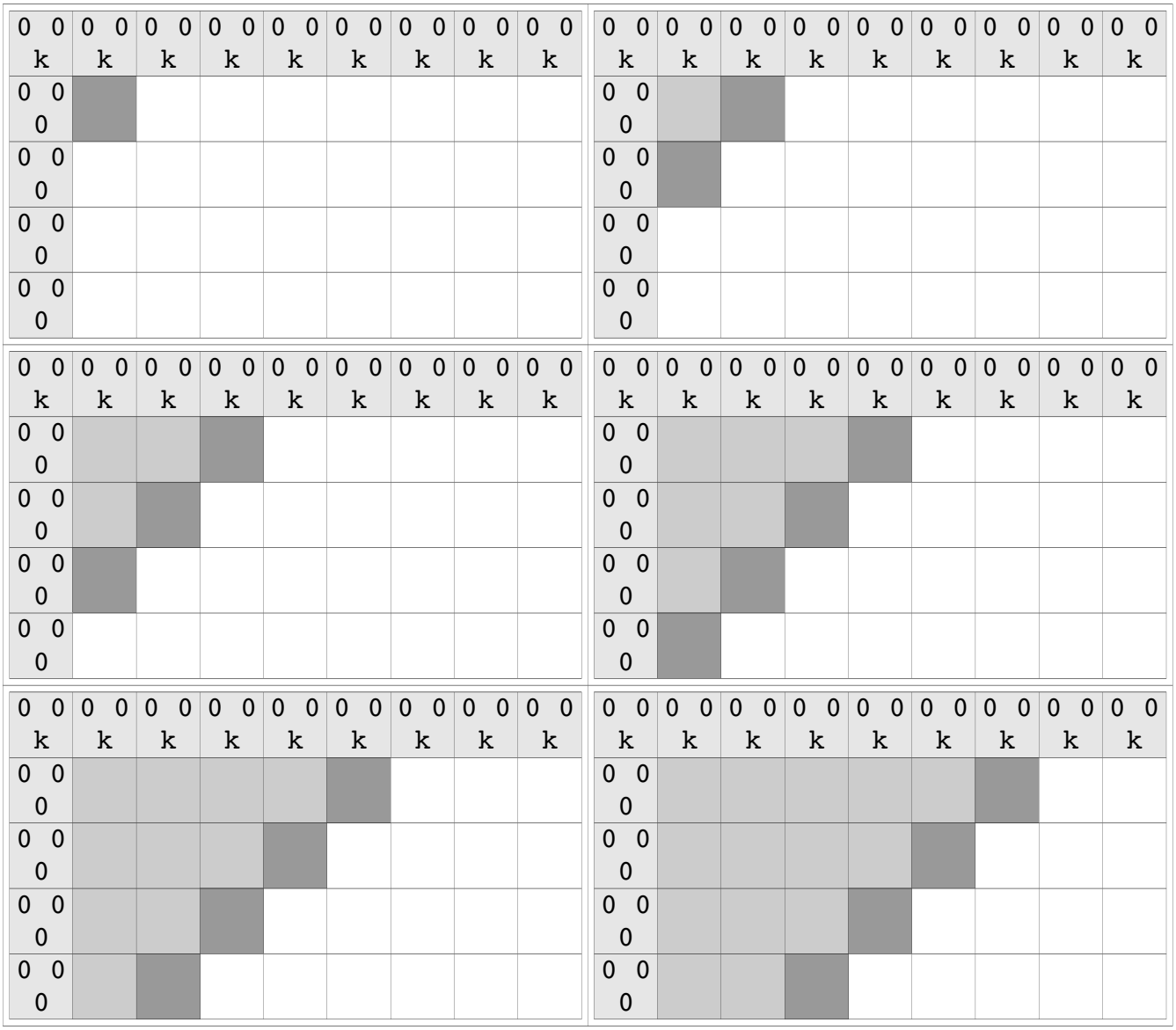
After having initialized the matrix, the remaining cells are computed using the following equations:

$$M_{r,c} = dist_{r,c}*[M_{\backslash}*MM(r) + X_{\backslash}*GapM(r) + Y_{\backslash}*GapM(r)]$$

$$X_{r,c} = M_{\uparrow} * MX(r) + X_{\uparrow} * XX(r)$$

$$Y_{r,c} = M_{\leftarrow} * MY(r) + Y_{\leftarrow} * YY(r)$$

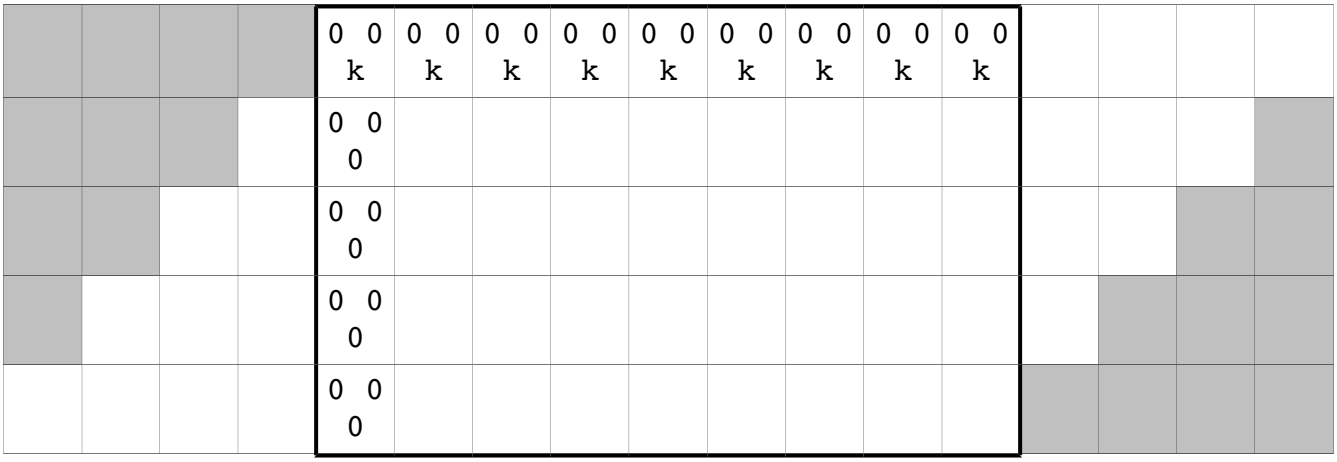
After we have initialized first row and column, many different ways can be choosen for the order in which the remaining cells of the matrices will be computed. The approach “by diagonals” has the advantage that there are no two cells in the same diagonal which depends on each other. In the calculations, the cells only depends on the 2 previous diagonals. This is an advantage mainly because it allows to exploit paralellism. An illustration of the first 6 diagonals computed after initialization follows.



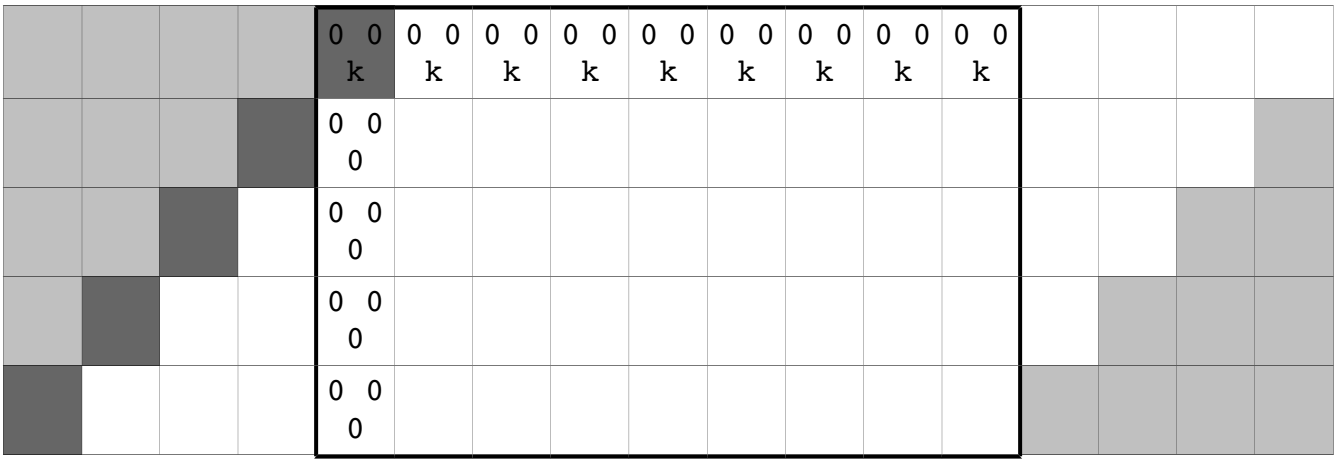
This approach is implemented in file pairhmm-4-hybrid-diagonal.cc.

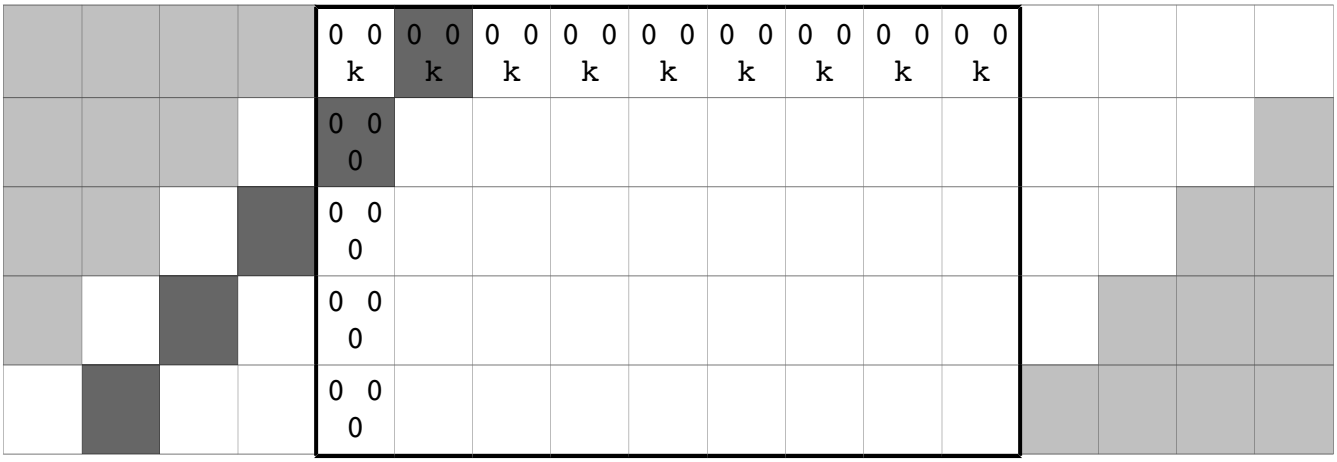
Notice that, altought the idea is “conceptually neat” (i.e.: iterate through all diagonals), at the implementation level it is not so neat, since the diagonals are not all of the same size; it becomes necessary to take this fact into consideration at the implementation level and solve it in some way. pairhmm-4-hybrid-diagonal.cc's iterates through all “potential” cells of the diagonal and checks if it IS a cell (i.e.: it checks if the cell is inside the matrix). If not, it just ignores the cell. The intention of the current commit is to make the computation exactly the same on every diagonal. The proposed way is to have a bigger matrix, with enough space to contain all the diagonals that are going to be accessed. In the following section we will discuss about the initialization issues.

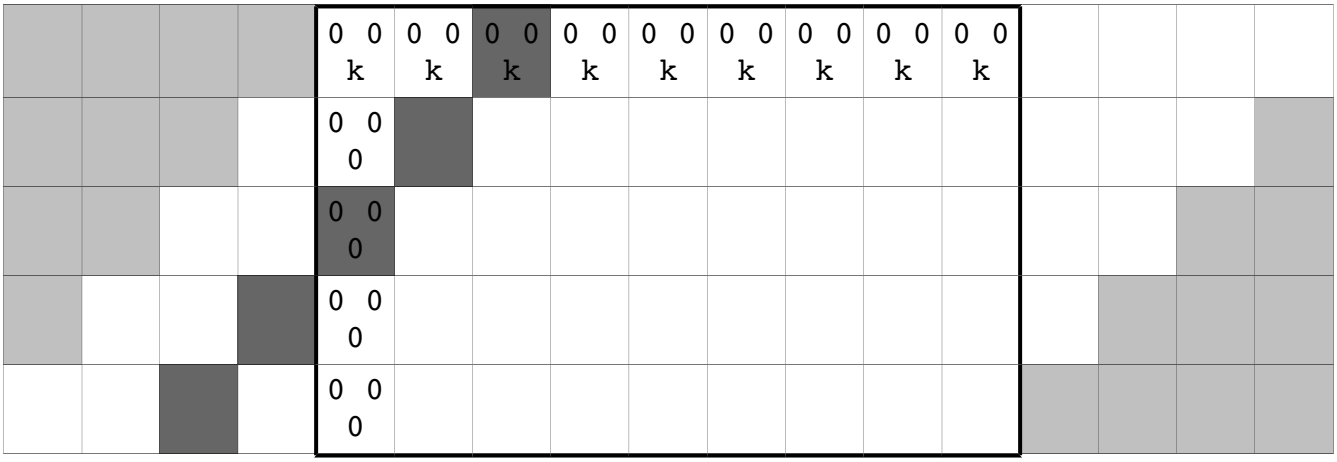
For the moment, let's notice that we want to have in memory a matrix like this (gray cells are never accessed, they are just allocated)



We will compute in the following order (just first 3 steps are shown)

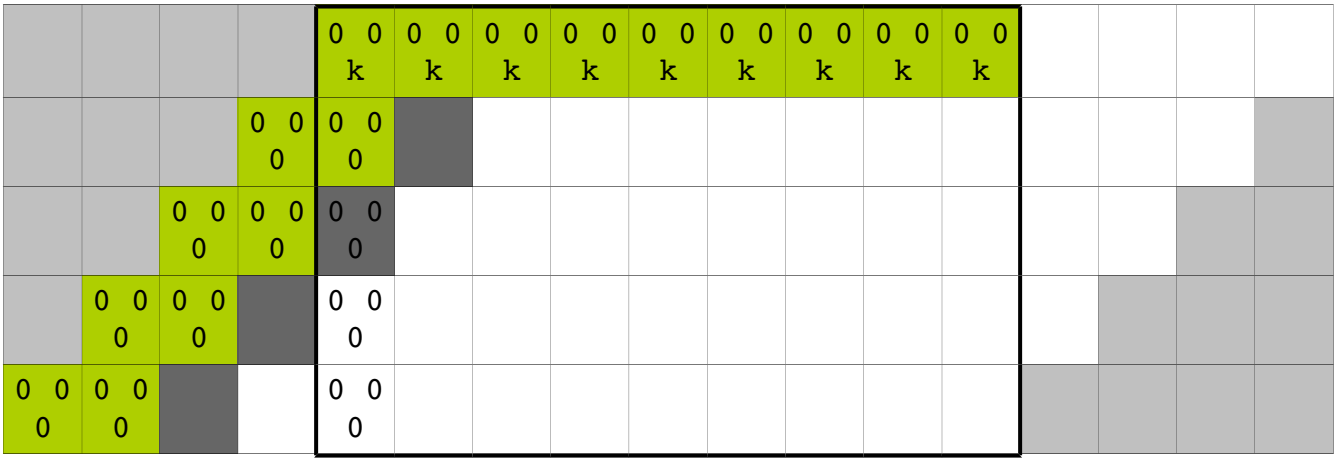






So, if we manage to initialize the diagonals in such a way that the computation of the first row and first column matches the initialization of those values in the previous approach, we will obtain the same values in the whole matrix.

We do in a simple way: we initializes the first row of every diagonal (and we keep the concept of not computing the first row, just initialize it), and we set to 0 all the values of the first two diagonals. As follows:



This is the approach implemented in the current commit.