The Optimization of Methods of Finding the Longest Common Subsequence and the Development of Algorithms of Their Implementation in a Graphical Processor

Grigoriev A.V., Manturov A. O., Terentiev A.A.,
Yuri Gagarin State Technical University of Saratov
E-mails: agrigorev@mirantis.com, manturovao@gmail.com, a_a_terentyev@mail.ru

Abstract

In the era of growing informatization of society more people deal with and more computing power is consumed by producing, storing and processing of information. One of the classic tasks of informatics is searching for the longest common sub-sequence of two and more sequences.

The task of finding the longest common sub-sequence is the task of searching for a sequence which is a sub-sequence of several sequences. A sub-sequence can be acquired from a certain finite sequence if a certain set of its elements (possibly, an empty set) is taken out of it. The array of tasks requiring finding the longest common sub-sequence is fairly large: finding the difference between files (diff) for a version control system (CVS), DNA sequencing, the task of finding plagiarism, solving multi-criteria problems with the help of genetic algorithm.

According to the latest research done by IDC and published in “Extracting Value from Chaos” [1], the volume of information across the world is doubled every two years, which is faster than what the Moore’s law states. It is therefore necessary to develop algorithms working in a correct and optimal way in order to process the information.

Dynamic programming

The technique of dynamic programming can be applied to produce global alignments via the Needleman-Wunsch algorithm, and local alignments via the Smith-Waterman algorithm. The dynamic programming method is guaranteed to find an optimal alignment given a particular scoring function; however, identifying a good scoring function is often an empirical rather than a theoretical matter. Although dynamic programming is extensible to more than two sequences, it is prohibitively slow for large numbers of or extremely long sequences. The technique of dynamic programming is theoretically applicable to any number of sequences; however, because it is computationally expensive in both time and memory, it is rarely used for more than three or four sequences in its most basic form. This method requires constructing the *n*-dimensional equivalent of the sequence matrix formed from two sequences, where *n* is the number of sequences in the query. Standard dynamic programming is first used on all pairs of query sequences and then the "alignment space" is filled in by considering possible matches or gaps at intermediate positions, eventually constructing an alignment essentially between each two-sequence alignment. Although this technique is computationally expensive, its guarantee of a global optimum solution is useful in cases where only a few sequences need to be aligned accurately.

Needleman-Wunsch algorithm

In Nidelman-Wunsch algorithm [2] a dynamic programming approach is using. Required to find the maximum one occurrence to another DNA sequence, the sequence length are n_1 and n_2 respectively. Suppose that solutions are already exist for all subtasks (m_1, m_2), the smaller the set one, where m_1 and m_2 - the length of the sequences such that $0 \le m_1 \le n_1$ and $0 \le m_2 \le n_2$. Then the problem (n_1, n_2) is reduced to a smaller subtasks by the follows:

$$substitute(n_{1,} n_2) = \begin{cases} 0, & n_1 = 0 \cup n_2 = 0 \\ subtitute(n_1 - 1, n_2 - 1) + 1, & s_1[n_1] = s_2[n_2] \\ max(substitute(n_1 - 1, n_2), subtitute(n_{1,} n_2 - 1)), & s_1[n_1] \neq s_2[n_2] \end{cases} \quad (1)$$

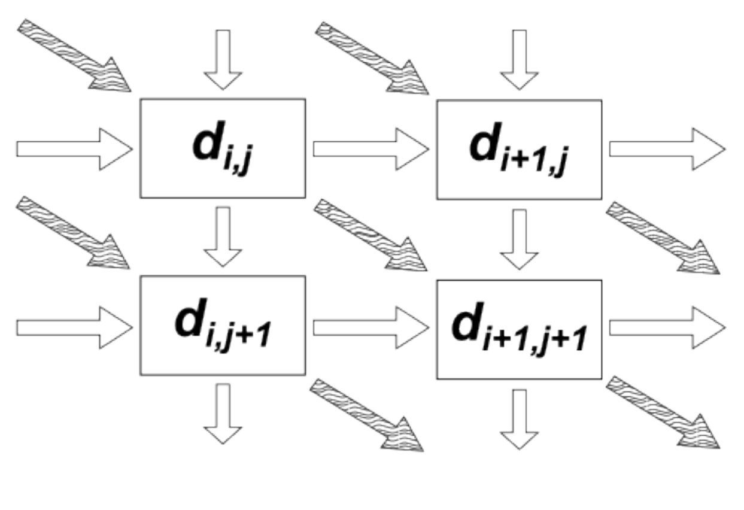

Fig. 1. Fragment of matrix M

Graphically, it can be represented as a matrix M (Fig. 1), which includes solutions subtasks (i, j) as its elements i, j .The arrows in the diagram are the transitions between subtasks solutions. Textured arrows indicate the transitions in which the length of the common subsequence increased, white arrows - transitions, which produce or extend breaks of one of the aligned sequences. In each cell d i, j of the matrix M maximum subsequences length is stored. The length can be obtained from first i elements of the first and j elements of second sequences. It is clear that the asymptotic estimate of the time of the algorithm is O ($n_1 * n_2$) [3], where n_1 and n_2 - the length of the DNA sequences. Since all the information is stored in a matrix, then the algorithm will occupy in memory capacity equal to $n_1 * n_2$.

The most economical on memory consumption (will hold of n_i) - is the Miller-Myers algorithm [4]. Its working principle is to break one of the sequences in two equal parts. For each section there is a line x weight (formula (1)), the optimal alignment of origin (0, 0) at the end of the x and (n_1, n_2) in the x: $W^+(x)$, $W^-(x)$. Weight optimal alignment passing through the point x (Fig. 2): W (x) = $W^+(x) + W^-(x)$. Weight optimal alignment is W = max_x (W(x)). So, we have found a point through which passes the optimal alignment. Found a point x breaks matrix M into four quadrants (submatrix M), two of which are not to contain the optimal alignment .For two quadrants containing the best way we can apply the same technique, and remember the point x 'and x ".Continuing the process of division in two quadrants, we find all the points through which the optimal path (Fig. 3).

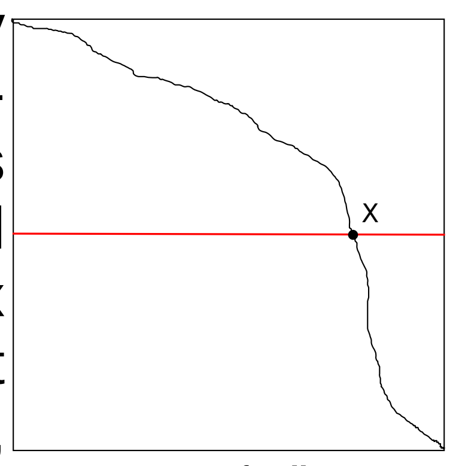

Fig 2. First step of Miller-Myers

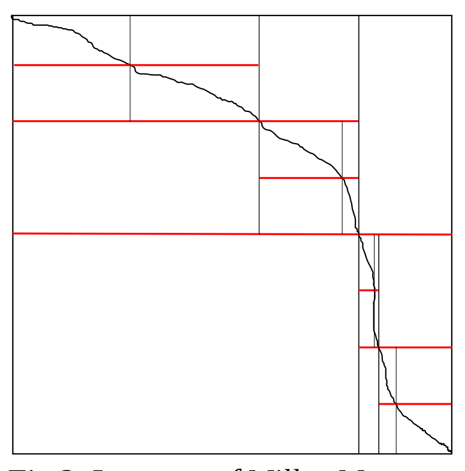

Fig 3. Last step of Miller-Myers

The algorithm for GPU

Modern graphics cards almost as good as supercomputing by speed and number of cores, but they have one very significant drawback: the graphical processor arithmetic units are grouped into blocks, and all the devices that are in the same block, perform the same operation. [5]This approach is ideally suitable for computer graphics, where you have to handle an array of pixels, or perform operations on matrix.

The Miller-Myers method will not be optimal to run on the graphics card. We can run calculation in one quadrant by 2 flow: the first do perform calculations from the beginning, the second - from the end of the matrix M and all operations for the calculations will be identical. In the parallel computation of several quadrants, we are faced with the following problem: there are usually have quadrants with different dimensions, and when one quadrant has already been calculated, the streams will have to wait to process the remaining quadrants. This reduces the performance of the algorithm as a whole. Need to parallelize computation within a quadrant-stand. This can be done by changing the order of computation each element. Usually sequential algorithm calculation of the matrix M is assumed[6]. We consistently go around the matrix line by line from top to bottom and by columns from left to right. The proposed algorithm will be carried out in an independent calculation of the matrix M elements on a line parallel to the secondary diagonal of the matrix. The above calculation procedure is shown in Fig. 4 on the right. It shows that the classical method (shown on the left) runs in 16 stages, and proposed an alternative algorithm – 7.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7

Fig. 4. Calculation order. Needleman-Wunsch on the left, alternatively on the right

Let's assemble calculation order and Miller-Myers main principle. We will split matrix M by lines parallel to the secondary diagonal of the matrix and find all points are belongs to solution. Fig. 5 and Fig. 6.

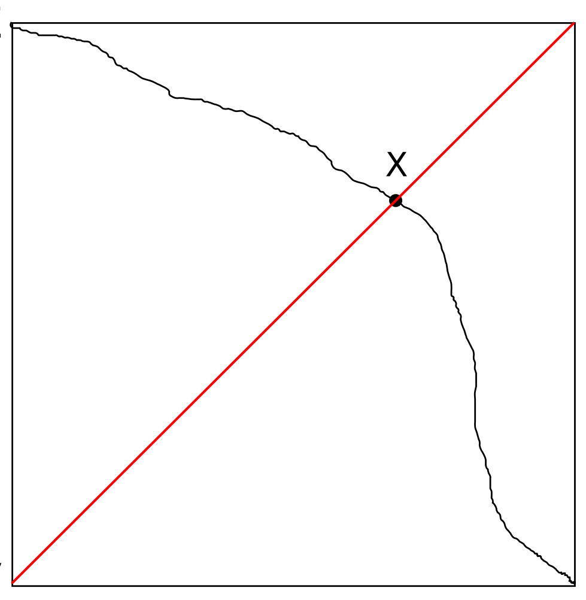

Fig. 5 Proposed algorithm. Step 1

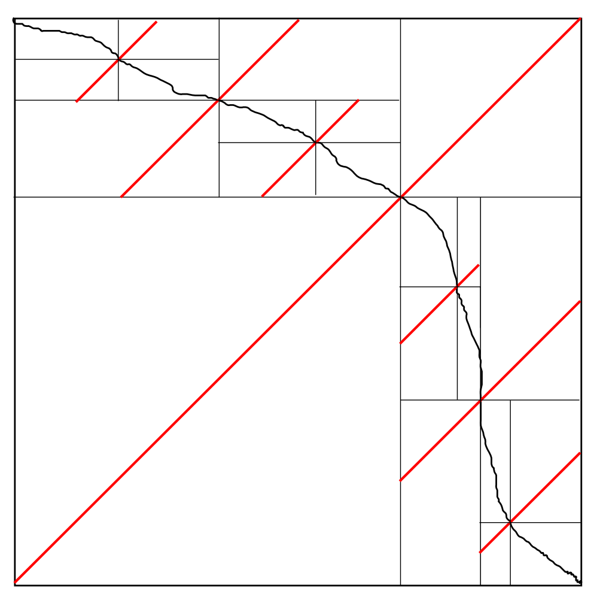

Fig. 6 Proposed algorithm. Last step

Bibliography

1. http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf
2. S.B. Needleman and C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, Journal of Molecular Biology, 48 (1970) 443-453.
3. Greene, D. H. and Knuth, D. E. *Mathematics for the Analysis of Algorithms, 3rd ed.* Boston, MA: Birkhäuser, 1990.
4. Jason Sanders, Edward Kandrot (2010) «CUDA by Example: An Introduction to General-Purpose GPU Programming» - Addison-Wesley Professional – 312
5. E.W. Myers and W. Miller, Optimal alignments in linear space, Computer Applications in the Biosciences, 4(1) (1988) 11-17.
6. http://www.ibm.com/developerworks/java/library/j-seqalign/index.html?ca=dgr-jw17dynamicjava&S_TACT=105AGX59&S_CMP=GR