Доклады Международной конференции "Математическая биология и биоинформатика". *Под ред. В.Д. Лахно. Том 7. Пущино: ИМПБ РАН, 2018. Статья № e64. doi: 10.17537/icmbb18.92*

Distributed realization of parallel algorithm for global alignment of long sequences

Pankratov A.N.[1, 2], Tetuev R.K.[1], Pyatkov M.I.[1]

[1]*Institite of Mathematical Problems of Biology RAS – the branch of Keldysh Institute of Applied Mathematics RAS*
[2]*Pushchino State Natural Science Institute*

pan@impb.ru

Distributed realization of parallel global alignment algorithm is proposed. The goal of this work is to collect the best practices and to build the most general model of a pairwise global alignment with the possibility of aligning as long sequences as possible. As a result, the versatile algorithm for global alignment is constructed on the basis of parallel implementation FastLSA of the Needleman–Wunsch algorithm with an arbitrary matrix of substitutions and Gotoh algorithm for the affine system of penalties for gaps. The main feature of the algorithm is low memory consumption. The algorithm is implemented in the Javascript programming language as web application. Distributed computing is implemented in the PHP programming language and is produced on clients that use this web service.

Key words: global alignment, affine penalty system, terminal inserts, distributed browser computing.

Распределенная реализация параллельного алгоритма глобального выравнивания протяженных последовательностей

Панкратов А.Н.[1, 2], Тетуев Р.К.[1], Пятков М.И.[1]

[1]*Институт математических проблем биологии РАН – филиал Института прикладной математики им. М.В. Келдыша РАН*
[2]*Пущинский государственный естественно-научный институт*

Предлагается распределенная реализация параллельного алгоритма глобального выравнивания. Цель этой работы заключается в сборе лучших практик и построении наиболее общей модели парного глобального выравнивания с возможностью выравнивания как можно более длинных последовательностей. В результате универсальный алгоритм глобального выравнивания построен на основе параллельной реализации FastLSA алгоритма Needleman–Wunsch'а с произвольной матрицей подстановок и алгоритмом Gotoh'а для аффинной системы штрафов за пробелы. Основной особенностью алгоритма является низкое потребление памяти. Алгоритм реализован на языке программирования Javascript как веб-приложение. Распределенные вычисления реализованы на языке программирования PHP и производятся на клиентах, которые используют данный веб-сервис.

Ключевые слова: *глобальное выравнивание, аффинная система штрафов, концевые вставки, распределенные браузерные вычисления.*

1. Introduction

In this paper, we consider the classical algorithm of bioinformatics – the Needleman–Wunsch algorithm [1] for the global alignment of nucleotide and amino acid sequences. The aim of this work is to develop a tool for assessing the similarity of extended repeats in genomes which can be detected by the generalized spectral-analytical method [2, 3].

Modern implementations of the global alignment algorithm such as corresponding services of NCBI BLAST and EMBOSS consider the affine penalties system for insertions proposed in the work [4], as well as the penalty system for insertions at the ends of sequences. At the same time, the parameters of the method in these implementations cannot always be changed within the desired limits, and the questions of choosing a unique alignment from a variety of alternative solutions are solved in different ways. Thus,

each implementation has its own characteristics which do not allow to compare it exactly with others. Moreover, the main limitation of these realizations is the length of the studied sequences being, as a rule, tens of thousands of nucleotides. This is since the algorithm of global alignment is quadratic in both memory and speed, depending on the length of the processed sequences.

Overcoming the memory complexity of the algorithm was made in the work [5], which proposed lowering the memory consumption to linear due to the recursive approach suggested earlier in the paper [6] for finding the common maximum length substring. The essence of this approach is that one of the sequences is divided in half and there is a corresponding point in the second sequence through which the optimal alignment passes. After that, the task is recursively reduced to aligning two pairs of fragments, etc. At the same time, the theoretical computational complexity is doubled in comparison with the classical algorithm in which the scoring matrix is stored in memory.

In this paper, the grid scheme of the algorithm, proposed in the work [7], is taken as a basis which allows both saving memory and parallelizing computations. In the grid scheme, the score matrix is divided into blocks, the memory is allocated to the values of the matrix elements at the block boundaries, and the values of the matrix elements inside the blocks can be calculated starting from the boundary ones. Calculations are saved by the fact that on the back pass of dynamic programming a small number of blocks are subject to recovery. It is shown that in the case of recursive application this approach also reduces the use of memory to linear.

However, unlike the works [5, 7] which proposed algorithms with linear memory growth, in this work the minimum of required memory is obtained as the balance between its use in the forward and reverse pass of the method without recursion. It is shown that, although memory growth in this case is greater than linear memory, it allows aligning sequences up to a million nucleotides. At the same time, the developed algorithm is characterized by a good scalability of computations on multi-core computing systems [8, 9].

2. Algorithm

Let us consider memory minimization strategy for FastLSA algorithm [7].

Suppose that L_1 and L_2 are the lengths of the sequences being compared, H_1 and H_2 are the grid steps for each sequence, respectively. Evaluation of the memory used by the algorithm contains two terms, the amount of memory on the forward and backward pass:

$$F(H_1, H_2) = C_1 \left(H_1 + H_2\right) \frac{L_1 L_2}{H_1 H_2} + C_2 H_1 H_2$$

The first term is obtained by multiplying the number of blocks of the grid $\frac{L_1 L_2}{H_1 H_2}$ by the number of stored boundary elements of each block $H_1 + H_2$ with coefficient C_1, which determines the amount of memory for one element of the boundary of matrix. The second term is proportional to the number of elements of one matrix block $H_1 H_2$ with coefficient C_2, which determines the amount of memory for one inner element of the matrix.

Constants in the current implementation are selected as follows:

$$C_1 = 3(8+1) = 27 \text{ and } C_2 = 3\frac{3}{8} = \frac{9}{8}.$$

On the forward pass the score value is encoded by 8 bytes. Additionally, one byte reserved for a vector of backward path which is encoded by 3 bits. It is multiplied by 3, the number of variants of scores and paths for the affine penalty system.

On the return pass, when the matrix is reconstructed inside the block, only a vector of paths must be stored because there is no need to store scores for internal elements. Additionally, in the current version, paths vectors are packed into the array of memory allocated for the matrix.

The minimum value of the above expression is equal

$$C_1^{2/3} C_2^{1/3} L_1^{2/3} L_2^{2/3}$$

what is found from the necessary extreme condition

$$\frac{dF}{dH_1} = 0, \quad \frac{dF}{dH_2} = 0$$

and is achieved at:

$$H_1 = H_2 = \left(\frac{C_1 L_1 L_2}{C_2} \right)^{1/3}$$

Note that even in the case of a rectangular matrix $L_1 \neq L_2$, the optimal size of the grid block is obtained to be the square $H_1 = H_2$. That is because the amount of memory used is proportional to the perimeter of the block. The optimal shape of the block is a square which contains the maximal number of inner elements with a minimal number of perimeter ones.

3. Distribution paradigm

The role of high-performance computing technologies in this task is to solve the problem in the initial formulation without introducing simplified versions of the algorithms.

The experience of constructing such algorithms testified that one of the strategies is the refusal to store intermediate data in favor of repetitive calculations with the opening possibility of parallelizing computations and minimizing the use of memory. For example, do not store the calculated values of the elements of some matrix, but have a way of calculating the elements of the matrix. As practice shows, this principle works both for vector-parallel computing systems with shared memory, and for systems with distributed memory [10].

When developing specialized software designed for a specific range of tasks and users, it becomes necessary to choose a platform for application development. Cloud computing technologies are the most promising both in terms of scalability and development of the

developed applications, as well as the convenience of providing access to the user.

Providing software to users in the form of web services is one of the most widespread and promising types of cloud technologies. At the same time, service developers are faced with the need to develop complex software and hardware systems that support these technologies. With the development of the service there is a need to expand its capabilities and attract more users.

While each user ponders his task, his computer is often idle. This computer time naturally through the browser's virtual machine can be given to help the server or other clients. Thus, a distributed computing system composed of service users can be built. This path can be chosen when building free services for scientific calculations. The problems of implementing such distributed computing systems have recently been actively discussed in the scientific literature [11–14].

For example, in the field of bioinformatics, scientific services on the Internet are often necessary in the biologist's work. However, the creation of such services, often associated with large amounts of information, requires special organizational conditions. Not solving this problem, it is possible to consider the construction of a computationally powerful but cheap service from the hardware point of view.

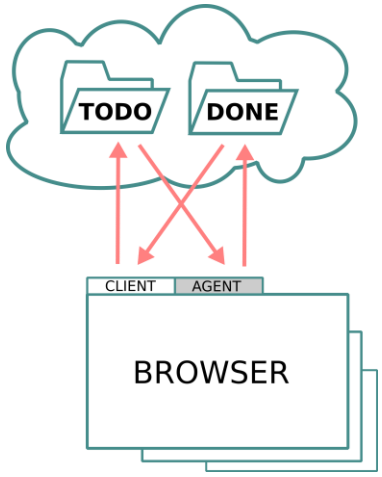

Fig. 1. Data transfer between clients, agents and server folders. Note that the client and the agent are just different browser bookmarks.

We propose the following simplified computing architecture, consisting of a server and two kinds of clients: master client, or simply client, and slave client, or agent (Fig. 1). In practice, it is assumed that two types of client are combined in one with two roles. The roles of these three agents of the computer network are strictly separated.

The master client plays the role of the main application in the system: it starts the computation process with the given parallel algorithm over the data that is also under its control. The master client divides the task into independent computational portions, generates tasks for them, sends tasks to the server, and accepts the resolved tasks from the server.

The server is a manager of packages, formed by master clients, and does not know anything about the tasks to be solved. It accepts packets with tasks from master clients, sends them to the slave clients for execution, takes the results back, returns the packets with the results to the master clients.

A slave client is a background computing process in the browser's virtual machine, which is in communication with the web server.

Thus, this architecture naturally unites users of the resource and socializes their computing power to solve computational problems using the specified software. Each client acts simultaneously as a master client, under the control of the user initiating the calculations, and an overhead slave client performing shadow tasks from other users.

This approach to the organization of distributed computing, which can be called a virtual cluster, has several advantages: voluntariness, mobility, cross-platform, security. The target is to increase memory and performance. Additional security, performance, and scalability issues need to be studied during further research.

3. Results

The table shows the optimal values of memory and block size calculated by the above formulas end execution time for nucleotide sequences of different lengths.

Table 1. Required memory, optimal grid step and execution time (estimated on 4 cores Intel Core i5 / MS Windows / Mozilla Firefox) for different lengths of sequences

$L_1 = L_2$	Memory, MBytes	$H_1 = H_2$	Execution time, s
10^5	130	6214	90
10^6	2808	28845	9000
10^7	60499	133877	–

Current versions of the service are published on the Internet at [15].

The program has the following options: an arbitrary matrix of substitutions, switching between three classic models of penalty system: constant, linear and affine; end gap penalties; selection of back-trace priorities, including random one; asymmetric penalty systems for each sequence; possibility of infinite penalty for insertion. The program aligns the sequences in any alphabet what is determined by the substitution matrix.

The user interface allows loading the sequence from the FASTA file and presetting the group settings of the program copying the default settings of the corresponding services.

Implementation means that the program does not require installation and runs locally on the client machine parallelizing the specified number of cores using web-worker technologies and using automatic grid step selection to minimize memory. The program was tested to align nucleotide and amino acid sequences up to the order of 10^6. For example, testing was carried

out on random nucleotide sequences that showed a 50 percent similarity regardless of the length at the default settings of algorithm.

4. Acknowledgments

This work was supported by the Russian Foundation for Basic Research [15-29-07063, 16-01-00692].

5. References

1. Needleman S., Wunsch C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 1970. V. 48. P. 443–453. doi: 10.1016/0022-2836(70)90057-4.
2. Pankratov A., Pyatkov M., Tetuev R., Nazipova N., Dedus F. Search for Extended Repeats in Genomes Based on the Spectral-Analytical Method. *Math. Biol. Bioinf.* 2012. V. 7. № 2. P. 476–492. doi: 10.17537/2012.7.476.
3. Pyatkov M., Pankratov A. SBARS: fast creation of dotplots for DNA sequences on different scales using GA-, GC- content. *Bioinformatics.* 2014. V. 30. № 12. P. 1765–1766. doi: 10.1093/bioinformatics/btu095.
4. Gotoh O. An improved algorithm for matching biological sequences. *Journal of Molecular Biology.* 1982. V. 162. P. 705–708. doi: 10.1016/0022-2836(82)90398-9.
5. Myers E., Miller W. Optimal alignments in linear space. *Computer Applications in the Biosciences (CABIOS).* 1988. V. 4. P. 11–17. doi: 10.1093/bioinformatics/4.1.11.
6. Hirschberg D. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*. 1975. V. 18. № 6. P. 341–343. doi: 10.1145/360825.360861.
7. Driga A., Lu P., Schaeffer J., Szafron D., Charter K., Parsons I. FastLSA: A Fast, Linear-Space, Parallel and Sequential Algorithm for Sequence Alignment. *Algorithmica.* 2006. V. 45. P. 337–375. doi: 10.1007/s00453-006-1217-y.
8. Galvez S., Diaz D., Hernandez P., Esteban F., Caballero J., Dorado G. Next-generation bioinformatics: using many-core processor architecture to develop a web service for sequence alignment. *Bioinformatics.* 2010. V. 26. P. 683–686. doi: 10.1093/bioinformatics/btq017.
9. Tetuev R., Pyatkov M., Pankratov A. Parallel algorithm for global alignment of long aminoacid and nucleotide sequences. *Math. Biol. Bioinf.* 2017. V. 12. P. 137–150. doi: 10.17537/2017.12.137.
10. Pankratov A., Tetuev R., Pyatkov M., Toigildin V., Popova N. Spectral analytical method of recognition of inexact repeats in character sequences. *Proceedings of the Institute for System Programming.* 2015. V. 27. № 6. P. 335–344. doi: 10.15514/ISPRAS-2015-27(6)-21.
11. Cushing R., Putra G., Koulouzis S., Belloum A., Bubak A., de Laat C. Distributed Computing on an Ensemble of Browsers. *IEEE Internet Computing.* 2013. V. 17. № 5. P. 54–61. doi: 10.1109/MIC.2013.3.
12. Pan Y., White J., Sun Y., Gray J. Gray Computing: A Framework for Computing with Background JavaScript Tasks. *IEEE Transactions on Software Engineering.* 2017. P. 1–1. doi: 10.1109/TSE.2017.2772812.
13. Fabisiak T., Danilecki A. Browser-Based Harnessing of Voluntary Computational Power. *Foundations of Computing and Decision Sciences.* 2017. V. 42. № 1. doi: 10.1515/fcds-2017-0001.
14. Zorrilla M., Florez J., Lafuente A., et al. SaW: Video Analysis in Social Media with Web-Based Mobile Grid Computing. *IEEE Transactions on Mobile Computing.* 2018. V. 17. № 6. P. 1442–1455. doi: 10.1109/TMC.2017.2766623.
15. *Aligner service project page*. URL: http://sbars.impb.ru/aligner.html (accessed 31.08.2018).