PARALLEL ALGORITHM FOR SEARCHING REGULATORY SIGNAL IN BACTERIAL GENOME

[1] ***Lyubetsky V.A., Rubanov L.I.***

Institute for Information Transmission Problems, RAS, Moscow, Russia, e-mail: lyubetsk@iitp.ru
[1] Corresponding author.

Key words: *informational genetics, regulatory signal, parallel computing, MPI*

Resume

Motivation: The paper describes a method of transforming existing sequential algorithm of regulatory signal search into a parallel version suitable for contemporary parallel computers supporting MPI protocol. This parallel implementation of the algorithm does not strictly bounded to the number of available processors and features a linear dependence of calculation speed on the number of processors.

Results: The algorithm allows biologists to investigate bigger sets of genomic sequences than they can study earlier. That has been proved in real experiments with the parallel algorithm carried out on teraflops parallel supercomputer at JSC of RAS.

Availability: The software is available on request from the authors.

Introduction

The problem of searching a regulatory signal in a set of assumed regulatory domains (sequences of nucleotides) is well-known. For its solution various sequential algorithms were proposed. In particular, the algorithm described in (Danilova, Gorbunov et al., 2001) was effective for many natural samples. To solve the problem, powerful computing systems sometimes are required that mean parallel computers in practice. The role of such systems in computational genomics is expected to increase because they enable considering tasks and data dimensions that are impossible to analyze by existing sequential architectures in principle. So it is desirable to consider possible ways and methods of transition from consecutive algorithms to their parallel versions using as example some problems important for genomics. The parallelization is known to be nontrivial in many cases. We succeed in the transition for several algorithms, and consider here this subject as applied to the mentioned consecutive algorithm. Specifically, we have developed effective parallel algorithm for searching regulatory signals. Detail results of its application to natural and artificial samples (including comparison with those of consecutive algorithms) are being published in the online magazine *Informational Processes* at http://www.jip.ru.

Difficulties of parallel implementation

Parallelization of the existing algorithm is complicated with two factors which are typical for many algorithms in genomics: cyclic (i.e. consecutive) structure of the algorithm and the model of the shared memory it is implicitly based on. The latter complicates interchanging data in the multiprocessor system. Actually, the majority of modern parallel computers divide random access memory more or less equally among processors interfacing each with other via internal network. The communication is normally accomplished by short messages conforming to e.g. MPI standard. Apparently, this method of data sharing is more time-consuming than direct access to the shared memory. Now we shall discuss the first of the mentioned difficulties.

Our sequential algorithm has a structure outlined in Fig. 1. Here each repetition of the loop seeks for quasioptimal solution of the problem for some fixed way of numbering initial nucleotide sequences. For brevity this way of numbering we call *permutation*, and process of the problem solving will be called *assembling* of the given permutation. Once a permutation has been assembled, the algorithm generates next permutation, and so on. The optimal solution would be the best one found from all possible $n!$ permutations, that is of course unattainable.

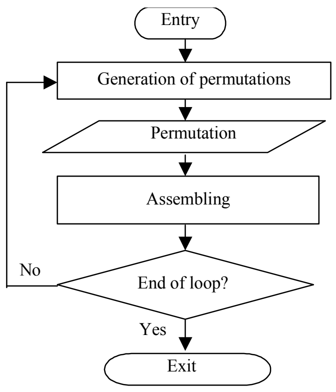

Fig. 1. Source algorithm.

Really we choose a termination condition of the loop so that the solution obtained from checked permutations would be close to absolute optimum in terms of some fixed functional of quality. For that we proposed two principles and two corresponding conditions of the algorithm termination. The first principle takes into account quality of the found signal and relative contribution of each sequence to joint quality: the more valuable site of the signal a sequence gives, the smaller serial number it has in the next permutation. Another principle takes into account completeness of covering the set of sequences by the principal edges of *G* tree which controls consecutive dichotomy of this set in that sequential algorithm.

The main burden of calculations (and, therefore, operating time of the algorithm) fall to the assembling phase. Since this block is located inside a loop (see Fig. 1), one can see the only opportunity to parallelize the algorithm: to perform parallel calculations inside the block. Though possible in principle, the assembling process is hardly scalable: vector-like data structures correspond to consecutive dichotomy of the set of source sequences. So dimension of a vector becomes rigidly connected to number *n* of sequences, and besides changeable at different levels of *G* tree, i.e. stages of assembling. In addition, time of computation for separate elements of such vector (i.e. sequence pairs) may vary over a wide range. But we need to wait until the slowest component complete prior to process the next level of the tree. Thus, an attempt to rigidly fasten separate stages or similar elements of assembling to processors would not be time effective, nor allow us to use all available processors and balance their workload. We come to necessity of essential changes in the very logic of consecutive algorithm.

Two-dimensional list of permutations and wavelike scheme of computation

When looking at algorithm (Danilova et al., 2001) as an optimization of given functional of quality, one can observe the following analogies. Each permutation is similar to a point in the space of optimum search, and assembling leads to the result similar to calculation of the functional value in this point. The consecutive algorithm is capable to generate points of two kinds corresponding to abovementioned principles. Namely, the second principle of coverage maximization provides new base points to search *global* optimum (we need them as we have no information on geometry of a response surface). And with the first principle we aspire to reach the best *local* solution, starting at the chosen base point.

The general idea of proposed parallelization method is that all available processors of the parallel computer system (except for one root processor) every moment are engaged to local optimization, each one starting at its own base point. All data necessary to apply the first principle are kept in local memory of the processor in charge. The dedicated root branch receives only results of local calculations (recognized signals along with their quality data). On completion of local optimization (this process we call *extension*), released processors receive new base permutations, and so forth. Termination criterion of this algorithm will be exhaustion of entire set of the base points generated with use of the second principle (maximum coverage), provided that all parallel branches finish their computations.

To implement this idea we shall generate not a straight queue of permutations like in consecutive algorithm, and the two-dimensional <*P, Q*> list (see Fig. 2). Its backbone is *P-list* of permutations $P_1, P_2, \ldots, P_k$ composed in the beginning of the algorithm; the length *k* of this list is determined by a desirable coverage degree of the set of base points. For example, if we demand that each pair of source genomic sequences appears at least once as principal edge at the upper level of *G* tree, i.e. these sequences fall into different halves at the very first dichotomy of entire set, then, obviously, $k = n\,(n-1)$:2.

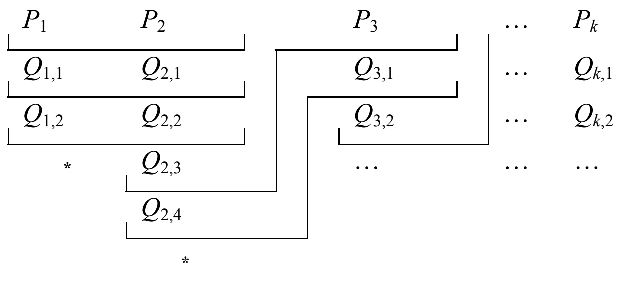

Fig. 2. <*P, Q*> list of permutations and assembling order.

Further, each element of the *P*-list initiates so called *Q-list* corresponding to the process of extension mentioned above. These *Q*-lists are being constructed dynamically during work of the algorithm. Specifically, after the analysis of quality of the signals already found in *i*-th *Q*-list (including its root P_i), the list either proceeds with a next permutation $Q_{i,j}$ or terminates (that is symbolically denoted by asterisk in Fig. 2). In the latter case the processor earlier serving *i*-th *Q*-list is released and takes the next unprocessed element of *P*-list, extending from it a new *Q*-list.

It is easy to see that in such two-dimensional set of permutations parallel processing is conducted like a computing wave propagating from the left top corner of the structure in a downward direction and to the right until whole <*P, Q*> list is processed. For the sake of simplicity Fig. 2 shows successive positions of the wave front for two processors working in parallel and with the assumption of constant duration of assembling any permutation, however one can easy imagine this algorithm in general case.

From the parallelization efficiency point of view the described scheme has many advantages from which we shall note three the most important: (1) it allows to put into operation at once the multitude of processors (at least as many as is the length *k* of *P*-list); (2) all secondary branches work on the same algorithm assembling their given permutations irrespectively of other branches; and (3) low-intensive data exchange between each of secondary branches and root branch of the algorithm: a permutation itself towards the secondary branch (*n* values), and found signal along with a quality of each its word (2*n* values) towards the root branch.

These advantages allowed us to create the parallel algorithm independent of any specific parallel computer and number of available processors; the only prerequisite is that the computer complex supports MPI standard (at least the first edition). We also avoid using shared memory due to move criteria checking and permutation generation to the single root branch. The only common data (a matrix of pair-wise proximities of all words from all sequences) is unchanged during work of the algorithm, so may be calculated and stored simultaneously in all parallel branches.

Implementation and Results

The described parallel algorithm of regulatory signal search has been implemented as 32 bit console application written in ANSI C. It may be compiled by e.g. gcc, pgcc, Borland C++ 5.02 compilers without any modification, and works in Windows 9x/NT4/ME/2k/ XP and Linux environments. Debugging and initial testing of the program were carried out on PCs with use of WMPI 1.54 software by Critical Software. Real experiments are being performed now on MBC1000M supercomputer in the Joint Supercomputer Centre of RAS et al. Results obtained for artificial and real examples show high efficiency of parallelization in the algorithm: busy time ratio of secondary processors equal 94-96%. Experiments also confirm theoretically predicted linear dependences of assembling time on number of source sequences (Fig. 4) and computation speed on number of available processors (Fig. 5). We could not recognize any tendency of deceleration this linear growth of performance.

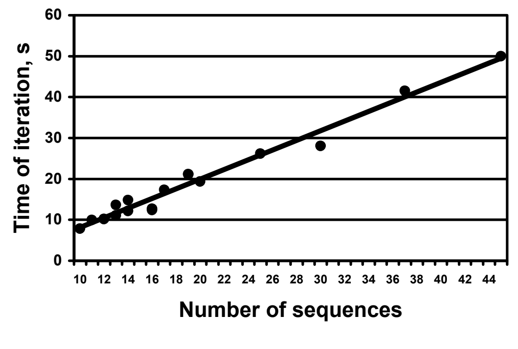

Fig. 4. Assembling time as a function of number of sequences.

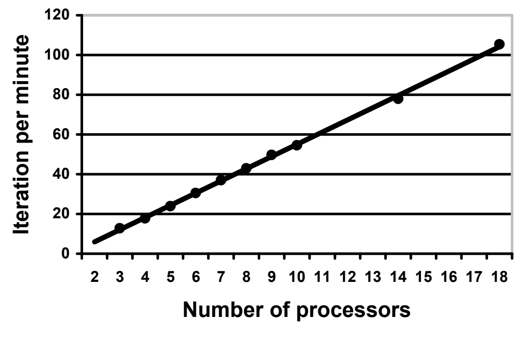

Fig. 5. Linear growth of parallel algorithm performance (n=14, m=200, l=20).

Reference

1. Danilova L.V., Gorbunov K.Yu., Gelfand M.S., Lyubetsky V.A. (2001) Algorithm for extraction of regulatory signals in DNA sequences (2). Mol. Biol. 35, 6, 987-995.