Computer Physics Communications 196 (2015) 390–397

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A parallel algorithm for random searches

M.E. Wosniack [a,*], E.P. Raposo [b], G.M. Viswanathan [c], M.G.E. da Luz [a]

[a] Departamento de Física, Universidade Federal do Paraná, C.P. 19044, 81531-980 Curitiba-PR, Brazil

[b] Laboratório de Física Teórica e Computacional, Departamento de Física, Universidade Federal de Pernambuco, 50670-901 Recife-PE, Brazil

[c] Department of Physics and National Institute of Science and Technology of Complex Systems, Universidade Federal do Rio Grande do Norte, 59078-900 Natal-RN, Brazil

ARTICLE INFO

Article history:
Received 6 May 2015
Received in revised form
24 July 2015
Accepted 25 July 2015
Available online 4 August 2015

Keywords:
Random search
Parallel random search
Parallel random walk
Lévy flights

ABSTRACT

We discuss a parallelization procedure for a two-dimensional random search of a single individual, a typical sequential process. To assure the same features of the sequential random search in the parallel version, we analyze the former spatial patterns of the encountered targets for different search strategies and densities of homogeneously distributed targets. We identify a lognormal tendency for the distribution of distances between consecutively detected targets. Then, by assigning the distinct mean and standard deviation of this distribution for each corresponding configuration in the parallel simulations (constituted by parallel random walkers), we are able to recover important statistical properties, e.g., the target detection efficiency, of the original problem. The proposed parallel approach presents a speedup of nearly one order of magnitude compared with the sequential implementation. This algorithm can be easily adapted to different instances, as searches in three dimensions. Its possible range of applicability covers problems in areas as diverse as automated computer searchers in high-capacity databases and animal foraging.

1. Introduction

Surely, computational simulations constitute a keystone tool for our scientific understanding of nature [1]. However, given the diversity and the potential complexity [2] of necessary (numerical) models to study distinct phenomena, the optimization of the underlying algorithms becomes, in many instances, a crucial aspect [3] in their viability.

Towards this optimization goal, parallel computing strategies [4] are particularly important. Usually, algorithms and codes that rely on sequential decisions are not easily parallelizable, like P-complete problems [5,6]. But in spite of that, some processes thought to be inherently sequential, like the depth-first search [7], have been solved in a parallel fashion thanks to a proper distribution of work between the processors [8]. Other highly sequential instances, as edge coloring [9] and the maximal independent set [10], also have found alternative parallel solutions. Further, parallelized algorithms may be constructed in a very different way from the sequential counterparts, like parallel genetic algorithms that have a super-linear performance when compared to their sequential versions [11]. A collection of multidisciplinary problems allowing parallel approaches can be found in [12].

The general random search problem consists of finding a competent strategy for the encounter of randomly located target sites that can only be detected in the limited vicinity of the searcher. Its possible range of applicability covers areas as diverse as automated computer searchers of registers in high-capacity databases [13], motion of binding enzymes or proteins along DNA [14], economics [15], operational research like hunt for submarines [16], and animal foraging [17,18]. In certain contexts, such as in animal foraging [18], the knowledge of the distribution of encountered targets provides an important way to characterize how the resources are exploited during the search (e.g., if in an efficient manner).

In dealing with random search through numerical models [17,18], one often faces difficulties concerning the long time it takes to obtain meaningful results, a consequence of the large number of averages required. Actually, in this area of research [17,18] the computational procedures are traditionally sequential: the random walker moves from target to target until some halt criterion is achieved. The question is then how to formulate exactly the same problem, nevertheless using a framework allowing parallelization (with a consequent reduction of computational resources and time). One possibility is instead to consider a single

* Corresponding author.
E-mail address: wosniack@fisica.ufpr.br (M.E. Wosniack).

http://dx.doi.org/10.1016/j.cpc.2015.07.014
0010-4655/

random walker looking for Q targets, to assume Q random walkers looking for only one target. But for both models to lead to equal results, a carefully constructed extra condition must be imposed to the latter version. This extra information – in the form of initial conditions – concerns the spatial pattern that arises from the targets detected by a sole searcher, like footprints left by the walker on the original distribution of targets during the random search. Our present parallelized solution to the problem is based precisely on this idea. In fact, in order to recover the same distribution of detected targets observed in the sequential search routine, hence yielding the same statistical properties, our parallel implementation deals with a particular choice for the initial coordinates of each independent parallel random walker. This set of initial coordinates is actually built (as detailed below) from the original sequential search problem distribution of distances between two consecutively detected targets. It is basically a lognormal curve, whose mean and standard deviation values fully depend on the search strategy and environment density considered.

As for the protocol technical implementation, the parallelization is accomplished using OpenMP directives in the original (sequential) C code [19]. In the serial code, we have identified which functions could be independently processed, and then have used parallel directives to split the work between the threads. Each thread is able to execute a set of instructions independently, namely, to access the environment (stored in the shared memory), and to proceed with its own random walk according to its private variables. The OpenMP approach has been chosen because its directives have a simple implementation and also provide a natural solution for the memory consistency problem that appears in the parallelization.

The paper is organized as follows. In Section 2 we give a very brief overview about parallelization of problems involving random walks. The features of typical sequential random searches (important for our goals) are described in Section 3. In Section 4 we detail the proposed parallel search algorithm and compare with results from the usual sequential simulation. Finally, few remarks and the conclusion are presented in Section 5.

2. Few examples of parallelization of random walks based algorithms

As we are going to show in the next sections, our parallel algorithm for the random search problem actually promotes a considerable speedup of the simulations runs. So, certainly it is a contribution of practical importance in the field. However, conceptually the method is likewise relevant. By including a dynamical constraint to a reformulation of the original process, we are able to make it amenable to a parallel construction. Therefore, we are adding a new example to the previously mentioned (and not so large) list of systems which are essentially sequential in character, but even then can be parallelized.

Before going into our specific problem, few comments about the parallelization of random walks in a broader perspective are in order next. Random walks have been important tools in Monte Carlo simulations. For instance, to achieve flat histograms to calculate the density of states, the Wang–Landau method employs independent random walkers for different energies [20,21]. It allows a proper numerical solution for larger systems, with the independent random walkers playing a crucial role in this respect. One possible parallelization of the protocol assigns one random walk for each available thread [22], using thousands of threads from the GPU. Nevertheless, the sampling over the energy landscape is not sufficiently homogeneous, thus being necessary to specify more random walkers to the lower energy regions. An OpenMP implementation [23] has been designed for the Ising model using the Wang–Landau method. Moreover, the parallelism can be explored in the solution of partial differential equations that use the Monte Carlo method [24] with multiple random walkers.

In graph theory, an important application of parallel random walks is the study of the cover time of a graph (i.e. the necessary time to visit all the nodes). It is known that for some graphs of size n the use of k random walkers (with $k \leq \log n$) can decrease the cover time by a factor of k [25]. However, the choice of the initial coordinates for the walkers influences both the cover and the hitting times of random regular graphs [26,27]. In this sense, there is an optimal choice of initial coordinates, which is dependent on the topology of the graph, and that minimizes both times. The $s-t$ connectivity problem, in which one has to determine if the vertices s and t are connected to a same component, also gains efficiency when several random walkers are initialized. In wireless networks, the use of multiple random walkers in search for a target reduces both the computing time and the network overhead [28].

Further, in model-checking algorithms, where one has to verify the correctness of a system implementation, parallel random walkers have been successfully employed to explore the states space looking for errors [29]. These independent random walks can explore more states in a fraction of the sequential time, since the revisits to the same location are decreased in such arrangement [30]. Some model-checking algorithms are designed with coordinate random walkers, that simulate properties of animal foraging. The BEE algorithm [31], inspired by the cooperative behavior in bee hives, allocates parallel random walkers in regions of errors that were previously identified and communicated by an exploring random walker. The process is similar to the scouting of idle working bees when one bee identifies a profitable flower patch and informs its location to the colony. Another model with biological inspiration uses ant robots that work in a parallel and decentralized fashion, interacting only locally to explore a landscape efficiently [32].

All these examples illustrate the great computational gain in developing parallel procedures for algorithms based on random walks and searches. The present contribution goes exactly in this direction.

3. Sequential search properties

In order to build a parallel version of the random search problem, we present in this section some properties of the sequential version of the code that are necessary for our parallel implementation. The sequential random search model is discussed in detail in Ref. [18].

The search space is a square region of size D with periodic boundary conditions. In this environment a given amount of targets is distributed in a homogeneous way, with average distance l_t between them. The value of l_t (measured in terms of the radius of vision r_v, see below) characterizes the density of targets: the larger (smaller) l_t, the lower (higher) the density. The targets are non-destructive, i.e. they can be detected an unlimited number of times during the search. The searcher is a random walker whose step lengths ℓ are taken independently from a probability density distribution $P(\ell)$ at a random direction. The interaction between the searcher and the targets is provided by the radius of vision r_v that defines the region around the searcher where the target can be detected (here we set $r_v = 1$). Along the step j of length ℓ_j the walker constantly looks for targets within a distance r_v. If no target is found, the searcher completes the full step. Otherwise, a target is detected and the step ℓ_j is truncated. This process is then repeated with a new step and direction taken until the stop (halt) condition for the simulation is reached. For the probability density function of the step length ℓ we consider a power-law $P(\ell) \sim \ell^{-\mu}$ ($1 < \mu \leq 3$) for $r_v < \ell < D$ and 0 otherwise. This power-law distribution corresponds to the long-distance limit of the family of α-stable Lévy distributions with index $\alpha = \mu - 1$ [18]. The