Hybrid parallelization of a multi-tree path search algorithm: Application to highly-flexible biomolecules

Alejandro N Estaña, Kevin Molloy, Marc Vaisset, Nathalie Sibille, Thierry Simeon, Pau N Bernadó, Juan Cortés

▶ **To cite this version:**

Alejandro N Estaña, Kevin Molloy, Marc Vaisset, Nathalie Sibille, Thierry Simeon, et al.. Hybrid parallelization of a multi-tree path search algorithm: Application to highly-flexible biomolecules. Parallel Computing, 2018, 77, pp.84-100. 10.1016/j.parco.2018.06.005 . hal-01823694v2

HAL Id: hal-01823694
https://laas.hal.science/hal-01823694v2

Submitted on 4 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Hybrid parallelization of a multi-tree path search algorithm: Application to highly-flexible biomolecules

Alejandro Estaña[a,b], Kevin Molloy[a], Marc Vaisset[a], Nathalie Sibille[b], Thierry Siméon[a], Pau Bernadó[b], Juan Cortés[a]

LAAS-CNRS, 7 Av. du Colonel Roche, BP 54200, 31031 Toulouse cedex 4, France

[a] *LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France*
[b] *Centre de Biochimie Structurale. INSERM U1054, CNRS UMR 5048, Université de Montpellier, France*

Abstract

The study of the conformational energy landscape of a molecule is essential for the understanding of its physicochemical properties. This requires the exploration of a continuous, high-dimensional space to identify the most probable conformations and the transition paths between them. The problem is computationally difficult, in particular for highly-flexible biomolecules such as Intrinsically Disordered Proteins (IDPs). In recent years, a robotics-inspired algorithm called Transition-based Rapidly-exploring Random Tree (TRRT) has been proposed to solve this problem, and has been shown to provide good results with small and middle-sized biomolecules. Aiming to treat larger systems, we propose a hybrid strategy for the efficient parallelization of a multi-tree variant of TRRT, called Multi-TRRT, enabling an efficient execution in (possibly large) computer clusters. The parallel algorithm uses OpenMP multi-threading for computation inside each multi-core processor and MPI to perform the communication between processors. Results show a near-linear speedup for a wide range of cluster configurations. Although the paper mainly deals with the application of the proposed parallel algorithm to the investigation of biomolecules, the explanations concerning the methods are general, aiming to inspire future work on the parallelization of related algorithms.

Email address: juan.cortes@laas.fr (Juan Cortés)

Keywords: High Performance Computing (HPC), Hybrid parallelization, Path planning algorithms, Molecular energy landscape exploration, Intrinsically Disordered Proteins (IDPs)

1. Introduction

Experimental methods to analyze structural, thermodynamic and kinetic properties of molecules are expensive and, in many cases, they do not provide the expected accuracy, especially for flexible biomolecules. Computational methods have been developed over the last decades to complement experiment. These methods explore the conformational energy landscape of the molecular system, using physical or statistical models, aiming to determine physicochemical properties. Many different approaches have been proposed (see for example [1, 2]), which can be classified into two main categories: deterministic approaches, headed by molecular dynamics (MD) simulations [3], and stochastic approaches, mostly based on Monte Carlo (MC) methods [4]. When applied to large molecules, these methods are highly computationally demanding. Thus, parallel computing is almost mandatory. Basic MD and MC methods are sequential processes, but parallel computation is usually applied at a lower level for energy evaluation (and derivatives). More sophisticated variants of these methods can be parallelized at a higher level, as will be discussed in Section 2.1.

In recent years, algorithms originally developed for path planning in robotics have been proposed as an alternative to classic approaches [5, 6, 7]. This work focuses on one of these algorithms, the Transition-based Rapidly-exploring Random Tree (TRRT) [8], which performs a randomized exploration of the conformational space aiming to find probable transition paths between stable states of a molecule. More precisely, we present a parallel implementation of a multi-tree variant of TRRT [9, 10], which we will call Multi-TRRT hereafter (the principles of TRRT and Multi-TRRT are reviewed in Section 3). TRRT is based on the Rapidly-exploring Random Tree (RRT) algorithm [11], a popular path planning algorithm that can tackle complex problems in high-dimensional spaces.

Although path planning problems in robotics are not very computationally expensive in general, several approaches have been proposed for the parallelization of RRT-like algorithms aiming to reduce computing time. A brief survey on parallel path planning algorithms is provided in Section 2.2.

Today, the majority of high-performance computing (HPC) systems are clusters of multi-core processors[1]. To take advantage of this architecture, a current trend is to develop hybrid parallelization strategies, combining shared memory parallelization within each multi-core processor and distributed memory parallelization between processors. In this work, we propose such a hybrid parallelization strategy for the Multi-TRRT algorithm (Section 4). For the implementation, we use the standard and widely-used Message Passing Interface (MPI)[2] for inter-processor communication and Open Multi-Processing (OpenMP)[3] for intra-processor work. To the best of our knowledge, this is the first time that a hybrid parallelization approach has been employed within a robotics-based path search/planning method.

The performance of the parallel Multi-TRRT algorithm is evaluated using several molecules of different sizes. Indeed, the size of the molecule is directly related to the computational requirements of the problem. Results presented in Section 5 show the performance of the parallel algorithm, which achieves a near-linear speedup for large molecules.

Although the focus of this paper is on the parallel algorithm itself rather than on the application to molecular modeling, we would like to highlight here its relevance for the study of highly-flexible biomolecules such as Intrinsically Disordered Proteins (IDPs). The study of IDPs is extremely challenging due to their large size and inherent plasticity [12]. However, it is of great importance due to the key roles these molecules play in numerous biological processes, some

[1]To avoid ambiguity between *nodes* of the exploration tree and *nodes* of the computer cluster, in this paper we will refer to each component of a cluster as a *computer* or as a *processor*.

[2]https://www.open-mpi.org/

[3]http://www.openmp.org/

of which are related to pathologies such as cancer or neurodegeneration [13, 14]. Contributing to a better understanding of these biomolecules is our main motivation for the development of the methods presented in this paper.

2. Related work

Modeling molecular systems is computationally intensive. This has motivated numerous initiatives to introduce parallelization within this domain. This section first provides a brief survey on the parallelization of molecular dynamics simulation methods, which are the most frequently used computational techniques to study biomolecules. Then, more closely related to our contribution, we present a concise review of previous work on parallel path planning algorithms originating from robotics.

2.1. Parallel molecular simulation methods

Molecular Dynamics (MD) [3] is the most widely-used technique for studying molecular movements. This method explores the conformational space of a molecular system by numerically solving Newton's equations of motion. In MD, roughly 90% of the computing time is consumed in the calculation of the forces between non-bonded atoms. Since the 1980s, efforts have been focused on the parallelization of this part of the algorithm aiming to speed-up MD simulations [15]. Research in this area, aiming to further improve parallel implementations of this family of methods, is still very active (e.g. [16, 17]). Nowadays, parallel programming tools such as OpenMP (for shared memory architectures), MPI (for distributed memory architectures) or Charm++[4] (running on multiprocessor machines with or without a shared memory) are used for parallelizing MD algorithms and are incorporated within many software packages. This is the case for instance of the popular software packages GROMACS [18] and NAMD [19].

[4] http://charm.cs.illinois.edu/research/charm

More sophisticated variants of MD methods enable parallelization at a higher level. Replica Exchange Molecular Dynamics (REMD) [20] is probably the most clear representative of these advanced methods. REMD consists of running n instances of the same molecular dynamics problem with different parameter settings across the different replicas. After several iterations, the configurations of different replicas are exchanged if a stochastic transition test succeeds. The n simulations can be run in parallel, only requiring communication for replica exchange. REMD has been shown to sample the conformational space much more efficiently than a basic MD thanks to its ability to escape from local minima traps [21].

As an alternative to MD, Monte Carlo (MC) methods sample the conformational space of a molecular system by generating states according to a Boltzmann distribution [1]. Although the stochastic sampling process performed by a basic MC method is sequential, several parallelization strategies have been developed. The most simple approach is to run multiple independent MC explorations in parallel [22], aiming to provide a more exhaustive sampling of the conformational space. Parallalization in this setting is simple, since it requires no inter-process communication. However, more sophisticated parallelization strategies can be more efficient [23, 24]. In particular, the method proposed by Strid [23] improves the sequential MC version by obtaining several draws from the posterior distribution doing multiple evaluations in parallel. Another advanced variant, the Parallel Tempering Monte Carlo scheme, similar to REMD, enhances sampling by performing exchanges between replicas running in parallel [25].

2.2. Parallel path planning algorithms

Path planning algorithms have been developed since the 1970s to compute robot movements [26]. In the last two decades, sampling-based algorithms [27] have become very popular thanks to their efficiency, generality and conceptual simplicity. In addition to robotics, they have been applied to problems in other areas such as computational biology [5, 6, 7]. The basic principle of these algorithms is to construct a graph or a tree that captures the topology of the

admissible (i.e. collision-free or energetically-feasible) regions of the search space by randomly sampling configurations and attempting local connections. The Multi-TRRT algorithm enhanced in this work belongs to this family of methods.

When applied to simple robot systems, path planning algorithms are computationally fast, and parallelized implementations are not required. However, parallel computing may provide significant performance gains when dealing with complex systems. Starting from seminal work on the parallelization of classical path planning algorithms [28], most efforts have been focused on shared-memory parallelization strategies. This is particularly relevant when using multi-core central processing units (CPUs) [29] or many-core graphics processing units (GPUs) [30, 31]. Several approaches have also been proposed for shared-memory systems, with the aim to enable a more general and large-scale parallelization (e.g. [32, 33, 34]). In this work, we propose to combine both shared-memory and distributed-memory strategies. To the best of our knowledge, this is the first work proposing such a hybrid approach in the context of path planning algorithms.

As for molecular simulation methods, parallelization can be done at different levels of the algorithm. In the case of path planning algorithms for robotics applications, collision detection is the most computationally expensive operation, and parallelization can be focused on this part [30, 31]. Nevertheless, the algorithms can also be parallelized at a higher level using diverse strategies, as explained below. This paper focuses on high-level parallelization of the Multi-TRRT algorithm, but combining it with lower-level parallelization for collision detection and/or energy evaluation could be an interesting direction for future work.

The best parallelization strategy to be adopted depends on the characteristics of each path planning algorithm. Some algorithms present an inherent parallelism whereby they can be easily subdivided into *embarrassingly parallel* processes. This is the case for the Probabilistic Road-Map (PRM) algorithm [35]. Using a basic parallelization strategy, based on a shared-memory programming paradigm, significant speed-up can be achieved by building the roadmap coop-

eratively across multiple processes [32]. Other algorithms, such as RRT and its variants, are more difficult to parallelize due to the intrinsic sequentiality of the tree construction process. Nevertheless, since RRT-based algorithms have been shown to be more efficient to solve some classes of problems compared to PRM-based methods, their parallelization has also been investigated in recent years.

The simplest parallelization strategy, known as the *OR Parallel paradigm*, can be applied to all types of randomized algorithms. The idea is to run in parallel several independent instances of the same sequential processes using different seeds for the initialization of the random sampling process. The first instance to reach the solution reports it and the other processes terminate. This method reduces the execution time by multiplying the chances of finding a solution. It can significantly accelerate solving problems that have a huge variability on running time. This strategy has been applied successfully to randomized algorithms using distributed-memory architecture [36, 37] and in shared-memory machines [38, 39].

A recent work [34] presents and compares the performance of three distributed-memory parallel versions of RRT: OR parallel RRT, Manager-Worker RRT and Distributed RRT. The Manager-Worker RRT approach uses a single processor to manage the tree construction, whereas the other processors perform the calculation of the most computationally expensive part of the tree expansion. Thus, a single copy of the tree is maintained in the memory of the master processor. In the Distributed RRT strategy, all the processes collectively build a single search tree. Each processor needs to update its own copy with information provided by the others. Results presented show that the Manager-Worker RRT and Distributed RRT achieves good performance improvements when solving constrained problems [34]. However, both approaches present drawbacks that hinder scalability when applied to very large computer clusters. For the Manager-Worker RRT approach, the manager process can rapidly become a bottleneck. For the Distributed RRT strategy, communication time increases with the number of processors, resulting in limited scalability. Several strategies

can be applied to improve the performance of the simple distributed-memory approaches mentioned above, such as the subdivision of the search space among processes and the implementation of efficient nearest neighbor search methods [40, 29].

More intricate strategies can be applied for the parallelization of algorithms that simultaneously construct several exploration trees. For instance, a *master-client scheme* was proposed to solve large-scale problems using a forest of random trees [33]. In a first stage, processes are run in parallel to build a set of trees that can be RRTs [11, 41] or Expansive Space Trees (ESTs) [42]. Then, a single process scheduler distributes work to link the trees among client processes. Another strategy consists of exchanging information between several processors running instances of the same problem (each processor builds its own tree) with different random seeds, aiming to find the shortest path more efficiently [43].

The work presented in this paper shares ideas with some of the aforementioned approaches. As in [33], the construction of several (independent) trees is parallelized. As in [40, 29], the space is subdivided to improve computational efficiency. Note that in our approach, space subdivision is implicit and evolves during execution, compared to an explicit and constant subdivision proposed in related work.

3. The Multi-TRRT algorithm

This section presents the main principles of the Multi-TRRT algorithm [9, 10] that we have parallelized. The pseudo-code of the overall algorithm is presented in Algorithm 1. It incorporates the parallelization explained in next section. In all the pseudo-code presented in this paper, brown lines correspond to OpenMP commands for shared-memory parallelization, and blue lines indicate functions involving MPI calls for distributed-memory parallelization. Red text is used to highlight parts of the code involving only the master processor.

Multi-TRRT is a multiple-tree variant of the TRRT algorithm [44, 8]. It explores a continuous cost space $\mathcal{C}$ (i.e. the conformational space of a molecule,

Algorithm 1: Parallel Multi-TRRT

input : the configuration space $\mathcal{C}$; the extension step-size δ;
the energy function $E : \mathcal{C} \to \mathbb{R}$; the set of initial configurations Q_{init}

output: the tree(s) $\mathscr{T}$

```
1  (𝒯, 𝒮) ← initProcesses(Q_init)
2  #pragma omp parallel (NumThreads)
3  while not stoppingCriteria () do
4      T_i ← chooseNextTreeToExpand()
5      q_rand ← sampleRandomConf(S_i)
6      q^i_near ← findNearestNeighbor(T_i, q_rand)
7      q_new ← extend(q^i_near, q_rand, E, δ)
8      if q_new ≠ null and
9      transitionTest(T_i, E(q^i_near), E(q_new)) then
10         #pragma omp critical (addNodeAndEdge)
11         addNewNodeAndEdge(T_i, q^i_near, q_new)
12         (T_j, q^j_near) ← findNearestNeighbor(𝒯, q_new)
13         if distance(q_new, q^j_near) ≤ δ then
14             T_i ← merge(T_i, T_j, q_new, q^j_near)
15             #pragma omp critical (Communication)
16             MPI_send(Master, q_new, i, j)
17     updateStructures(q_new, S_i)
18 return(𝒯)
```

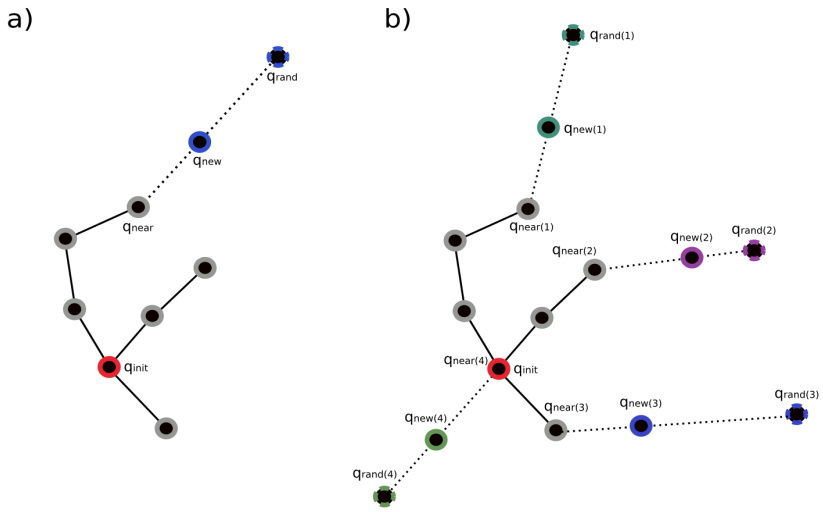

Figure 1: Illustration of the TRRT expansion process (for a single tree). The red node is the initial configuration and grey nodes have been generated in previous iterations. A new node q_{new} is created by the extension of the nearest node in the tree q_{near} toward a randomly sampled configuration q_{rand}. a) Mono-thread extension. b) Multi-thread extension, where k new nodes are generated in parallel.

in the present application context) by iteratively expanding several trees rooted at a given set of m initial configurations $Q_{\text{init}} = \{q^1_{\text{init}} \dots q^m_{\text{init}}\}$. At each iteration, a tree $\mathcal{T}_i$ is selected for expansion (line 4 in Algorithm 1). Following the principle of TRRT, the tree expansion involves three main stages, corresponding to lines 5-7 in Algorithm 1: 1) A configuration q_{rand} is randomly sampled[5]. 2) The nearest neighbor $q^i_{\text{near}} \in \mathcal{T}_i$ to q_{rand} is selected for expansion. 3) A new node q_{new} (and its corresponding edge) is created by extending/perturbing q_{near} in the direction of q_{rand} using a given step size. The principle is illustrated in Figure 1.a. This simple exploration strategy favors a rapid growth of the tree towards unexplored regions and guarantees convergence towards a uniform coverage of the reachable regions of $\mathcal{C}$ from q_{init} [44].

[5]In this pseudo-code, corresponding to the parallel implementation, q_{rand} is not sampled in the whole space $\mathcal{C}$, as is the case for the basic Multi-TRRT algorithm, but in a subset S_i. This will be further explained in Section 4.

A stochastic transition test (lines 9 in Algorithm 1) is then applied to q_{new} aiming to favor the exploration of low-cost/high-quality regions of $\mathcal{C}$. This important component of the TRRT algorithm will be further explained below. If the transition test succeeds, the new configuration q_{new} is added to $\mathcal{T}_i$ and connected to q^i_{near} (lines 10-11 in Algorithm 1). Then, the algorithm searches for the closest configuration q^j_{near} to q_{new} in all the other trees $\mathcal{T}_j \neq \mathcal{T}_i$. If q^j_{near} and q_{new} are close enough and the transition test is accepted in at least one direction, the two trees are connected (lines 12-14 in Algorithm 1). The exploration continues until all trees are merged or another stop condition (e.g. maximum number of iterations, timeout, ...) is reached. Figure 2 illustrates the behavior of TRRT (the single-tree variant is illustrated here for clarity purposes) exploring the conformational energy landscape of a very simple molecular system, alanine dipeptide [8]. The figure shows that the tree tends to explore low energy regions aiming to find the most favorable transition pathway connecting two given configurations.

The stochastic transition test applied to q_{new} (lines 9 in Algorithm 1) is based on the evaluation of a function $E : \mathcal{C} \to \mathbb{R}$ that associates a real-value cost with each configuration q. In the case of the molecular models considered in this work, the cost/quality function E correspond to the potential energy computed from a classical molecular mechanics forcefield [45], as generally used in molecular simulations. More detailed explanations about this cost/energy function are out of the scope of this paper. However, it is important to note that this energy function is computationally expensive, and typically scales quadratically with the size of the molecule being evaluated. The transition test within TRRT is inspired by the Metropolis criterion commonly used in MC methods, and involves a self-adaptive parameter T that we call temperature by analogy with methods in statistical physics. The pseudo-code is provided in Algorithm 2. Moves to lower-cost configurations are always accepted, and the temperature is unchanged in that case. Uphill moves are accepted with a probability that decreases exponentially with the local energy variation. After each accepted uphill move, T is decreased to avoid over-exploring high-cost regions. After

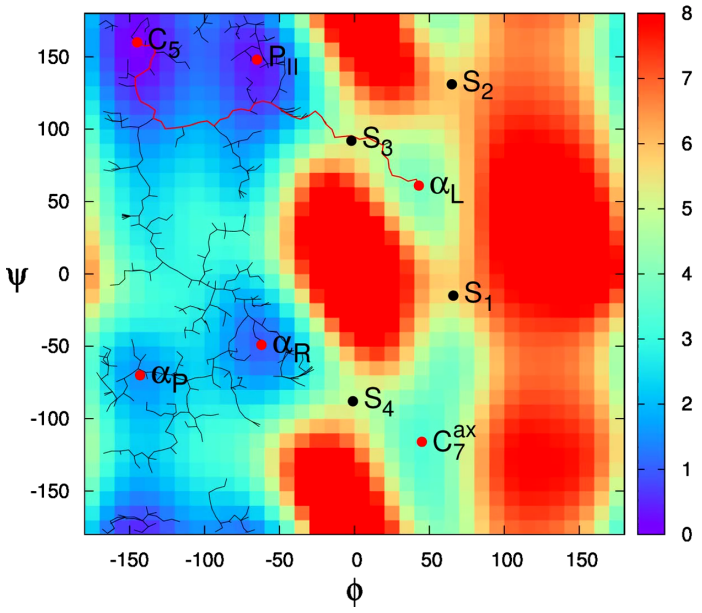

Figure 2: Illustration of TRRT exploring an energy landscape. The background image represents a two-dimensional projection of the conformational space of a small peptide. ssh aestana@olympe.calmip.univ-toulouse.frThe red dots represent the local energy minima (C_5, P_{II}, α_R, α_P, α_L, C_7^{ax}), and the black dots represent the main transition states (S_1, S_2, S_3, S_4). The background colors represent energy values with respect to the global energy minium, C_5. TRRT is applied here to find a path from C_5 to one of the local energy minima, α_L. The TRRT search tree rooted at C_5 grows on low-energy regions and explores other basins of the landscape before finding a higher-energy saddle region (around S_3) from which α_L can be easily reached.

each rejected uphill move, T is increased to facilitate the exploration and to avoid being trapped in local minima. The greediness of the algorithm depends on the parameter T_{rate}, which is a real value in the interval $(0, 1]$ provided as input. In general, $T_{\text{rate}} = 0.1$ has been empirically shown to be a good tradeoff between path quality and computing time [46].

4. Hybrid parallelization of Multi-TRRT

This section presents first a global overview of the approach, and then provides details of the parallel implementation.

Algorithm 2: transitionTest ($\mathcal{T}$, E_i, E_j)

input : the current temperature T; the temperature increase rate T_rate; the Boltzmann constant K

output: *True* if the transition is accepted, *False* otherwise

```
1 if E_j ≤ E_i then return True
2 if e^{-(E_j - E_i) / (K·T)} > 0.5 then
3     #pragma omp critical (Temperature)
4     T ← T / 2^{(E_j - E_i) / energyRange(T)}
5     return True
6 else
7     #pragma omp critical (Temperature)
8     T ← T · 2^{T_rate}; return False
```

4.1. General principle

The hybrid parallelization of Multi-TRRT presented below is aimed to better exploit current (multi-core) computer clusters. The idea is to minimize inter-processor communication overhead while taking advantages of shared-memory operations. In addition, this kind of parallelization permits easy adaptation of the algorithm to all types of computer architectures. For the implementation, we use a combination of OpenMP and MPI, which are standard and widely-used parallel computing tools. Some rules have to be respected between these two portable APIs in order to perform an efficient global interaction, as will be explained below.

A logical view of the architecture consists of p processes and k threads within each process. We consider one process per processor and one thread per physical core. In other words, we do not use the available hyper-threading technology that allows two threads per core. Although this technology is compatible with the proposed approach, we have not used it in this work to facilite the performance analysis of the parallel algorithm on the basis of physical cores rather than virtual cores. One of the processors is chosen as the master processor, which

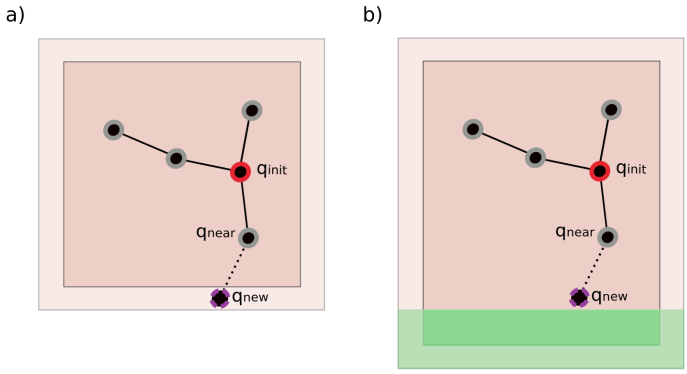

Figure 3: Illustration of the concept of a bounding n-polytope of a tree and its growth process. a) A new node is created near the boundary of the n-polytope, represented by the light brown region. b) The n-polytope is extended around the node (new green region).

will coordinate a few specific tasks while also performing the same task as the other processors. In the first stage of the algorithm (function `initProcesses` in Algorithm 1), the master processor distributes the m initial configurations Q_{init} among the processes. We consider $m \geqslant p$, so that each process builds m/p trees on average, and each tree building process is done almost completely independently (without the need for inter-process communication). The distribution takes into account the distances between the initial configurations in such a way that neighboring configurations are assigned to the same processor. This is important to reduce inter-processor communication during the multi-tree construction, as will be better understood with the explanations in Section 4.3.

An adaptive space subdivision approach is applied to avoid redundancies in the exploration performed by the different processes, and to reduce the communication between processors. The idea is to associate a bounding volume S_i, defined by a n-dimensional polytope (n-polytope), for each tree $\mathcal{T}_i$. The shape of the n-polytope evolves as the tree grows. If the n-polytopes S_i and S_j associated with two trees $\mathcal{T}_i$ and $\mathcal{T}_j$ do not intersect, this means that the exploration is taking place in different regions of the space. Thus, the trees can grow independently from each other, with no communication requirements between the corresponding processes. On the other hand, when two n-polytopes intersect,

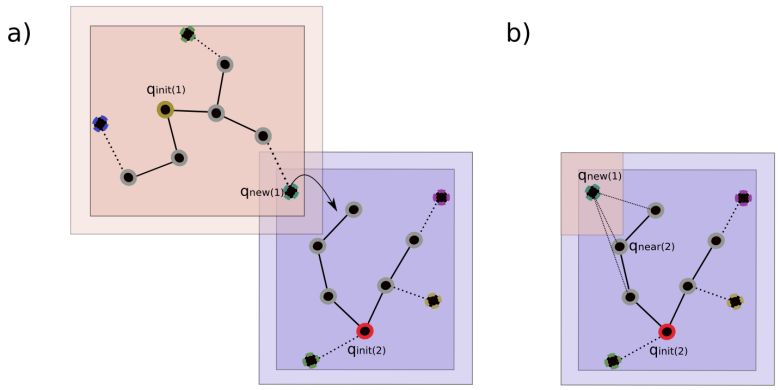

Figure 4: Illustration of the tree junction process. a) The extension of $\mathcal{T}_1$ generates a new node $q_{\text{new}(1)}$ in the intersection of bounding n-polytopes $\mathcal{I}_{1,2}$. b) The processor in charge of the construction of $\mathcal{T}_2$ tries to connect $q_{\text{new}(1)}$ to the nearest neighbor.

communication is required for attempting the connection of the associated trees. As will be further explained below, in our implementation, the master processor manages the information concerning the n-polytopes of all the trees, computes the intersections, and informs the other processors when they need to exchange data. The master process also detects when all the trees are connected, and informs that the exploration can be stopped.

4.2. Cooperative construction of trees inside each process (OpenMP)

Groups of trees are built using multiple threads within a shared-memory, multi-core processor. The idea is to parallelize the whole exploration loop. From the directive `#pragma omp parallel` (line 2 in Algorithm 1), several threads are created. The set of trees that are assigned to a processor are cooperatively constructed using all of that processor's threads (see Figure 1.b for an illustration). Each thread performs independently the tree selection, sampling and extension operations. The only operations that are required to be inside critical sections (i.e. cannot be executed in parallel) involve the modification of shared variables: the trees $\mathcal{T}_i$, the bounding n-polytopes S_i, and the temperatures T_i. Thanks to a careful implementation, the workload in critical sections is very small compared

Algorithm 3: updateStructures (q_{new}, S_i)

```
1  if q_new ≠ null and nodeNearBoundary(q_new, S_i) then
2      #pragma omp critical (updateNPolytope)
3      S_i ← updateNPolytope(q_new, S_i)
4      #pragma omp critical (Communication)
5      MPI_send(Master, S_i)
6      forall (I_{i,j}) do
7          if nodeInsideIntersection(q_new, I_{i,j}) then
8              #pragma omp critical (Communication)
9              MPI_send(Processor(j), q_new)
10 while MPI_received(q_new) do
11     (T_j, q^j_near) ← findNearestNeighbor(T, q_new)
12     if distance(q_new, q^j_near) ≤ δ then
13         MPI_send(Master, q_new, i, j)
14 while MPI_received(I_{i,j}) do
15     addToIntersectionList(I_{i,j})
16 if Master then
17     while MPI_received(S_j) do
18         if intersection(S_i, S_j) then
19             MPI_send(Processor(i,j), I_{i,j})
20     while MPI_received(q_new, i, j) do
21         updateGraphOfTrees(q_new, i, j)
22     if numberCC(GraphOfTrees) = 1 then
23         MPI_broadcast(endMessage)
```

to the rest of the process, which is essential for good performance. The most important implementation details are provided next.

Copies of the molecular system. Most of the operations concerning sampling and tree expansion require handling a model of the molecular system. Thus, a copy

of this model is provided to each thread in order to avoid race conditions. The drawback of this solution is that it requires more memory space. Nevertheless, this is not an important issue, since the memory space required by the copies of the molecular system is small compared to the space required by the exploration trees being constructed.

Multiple temperatures. In the basic Multi-TRRT algorithm, the temperature T is a global variable. In other words, a single variable T is used for the construction of all the trees. However, as each tree is exploring a different region of the space, it seems reasonable to assign different temperatures to different trees. This has several advantages. First, since different threads in a process are often working with different trees (especially when $m \gg p$), they are rarely blocked because of the critical section for updating the value of T. Besides, when using several processors, they do not need to communicate about variations of T, since it remains as a local variable for each tree. Furthermore, considering multiple temperatures improves the quality of the overall exploration (experimental results showing this are not presented in this paper).

Multi-threading nearest neighbor search. Efficiently locating the nearest neighbor is a difficult problem when considering large search trees and high dimensional configuration spaces, such as those of protein systems. In this work, we employ the Hierarchical K-Means Tree method within the Fast Library for Approximate Nearest Neighbors (FLANN) [47] (inside `findNearestNeighbor` function in Algorithm 1). The search can be executed simultaneously by multiple threads. However, the addition of a node to a tree requires single threading (a critical section) since the data-structure used for the rapid search has to be updated, requiring both reading and writing threads to block. To avoid a possible bottleneck, node insertions are first appended to a shared container (i.e. a buffer). The actual insertion is performed when the tree is not being actively accessed by any threads. In order to ensure a regular updating of the tree data-structure (protect against starvation), a size limit is enforced on the

container. When this size limit is exceeded, one of the threads enforces node insertion, blocking the access to the other threads.

Merging trees. When two trees constructed by the same process can be connected, they are merged into a single tree (lines 14-16 in Algorithm 1). The merging operation is performed by the thread that created the connecting node. To avoid race conditions during this operation, merging has to be delayed until all the other threads finish ongoing operations within the trees being merged. The lowest current value of the temperature T among the two trees is associated with the merged tree.

4.3. Limiting communication between processes (MPI)

The efficiency of a parallel algorithm strongly depends on the computational requirements associated with inter-process communication. To reduce communication requirements, we have implemented an adaptive space subdivision approach as explained below. The idea is to delay the communication between processes until it is necessary, and to use a master process to manage information exchange. The main operations requiring communication are performed inside the `updateStructures` function (line 17 in Algorithm 1), which is detailed in Algorithm 3.

Adaptive space subdivision. At the initialization (line 1 in Algorithm 1), a small n-dimensional polytope (n-polytope) S_i is associated with each $q^i_{\text{init}} \in Q_{\text{init}}$, where n is the dimension of the space being explored. In the current implementation, we use hyperrectangles because of the simplicity to update their shape and to compute intersections, acknowledging that other more accurate representations could be used. At the beginning, S_i is symmetric and centered on q^i_{init}. Random sampling for the expansion of each tree is performed in its corresponding n-polytope. When a new node is created near the boundary, the n-polytope grows in the direction of the tree expansion (lines 1-3 in Algorithm 3). The process is illustrated in Figure 3. This operation involves an OpenMP critical

section to avoid several threads trying to update the same n-polytope simultanously. Every time that a n-polytope is updated, the information is sent to the master processor (lines 4-5 in Algorithm 3), which then computes possible intersections with the n-polytopes associated with those built by other processors. Note that communication must be in an OpenMP critical section.

N-polytope intersection and trees connection. The master processor computes the intersection between the n-polytope S_i sent by a processor i and the n-polytopes S_j associated with trees managed by other processors (lines 17 in Algorithm 3). If the intersection $\mathcal{I}_{i,j}$ is not empty, the information is sent to the corresponding processors (lines 18-19 in Algorithm 3), and they add $\mathcal{I}_{i,j}$ to their n-polytope intersections lists (lines 14-15 in Algorithm 3).

When a processor i creates a new node q_{new} lying inside an intersection between n-polytopes $\mathcal{I}_{i,j}$, the node is sent to the corresponding processor j (lines 6-9 in Algorithm 3). Then, the other processor will try to connect the two trees (lines 10-13 in Algorithm 3), as in the basic Multi-TRRT. The process is illustrated in Figure 4.

Stopping condition and path extraction. In addition to performing space intersections, the master processor maintains a graph data structure to represent the connectivity between all the trees (lines 19-20 in Algorithm 3) using information sent by all the processors. The exploration process continues until a solution to the path finding problem is found, or if a stop condition based on a maximum number of iterations is satisfied first. The master process determines that a solution has been found when the connectivity graph contains a single connected component (lines 22-23 in Algorithm 3). Then, the solution path connecting all the initial configurations Q_{init} can be extracted. As the solution path is distributed among the processors, each processor i extracts the part of the path that connects q^i_{init} with the nodes that served as connectors with trees constructed by other processors.

4.4. Implementation framework

To implement the two aforementioned levels of parallelization, we use a combination of MPI (for distributed-memory parallelization) and OpenMP (for shared-memory parallelization). In such a hybrid framework, multiple threads may concurrently call MPI functions, requiring the MPI implementation to be thread-safe. Our algorithm makes MPI calls from all threads. In principle, two levels of thread safety (among the four available) can be used in our case: MPI_THREAD_SERIALIZED and MPI_THREAD_MULTIPLE. However, due to technical constrains of our implementation framework[6], we use the MPI_THREAD_SERIALIZED safety level. It allows multiple threads to make MPI calls, but not simultaneously, as is the case for MPI_THREAD_MULTIPLE. This implies that all the communications have to be performed inside OpenMP critical sections. Non-blocking receive operations (`MPI_irecv`) are used to reduce the time spent inside these critical sections.

Since our implementation is constructed in C++, we required the C++ MPI bindings. In addition, our application messages may contain instances of high-level classes, whose attributes can be pointers or Standard Template Library (STL) containers, which is incompatible with low-level MPI communication, requiring the programmer to explicitly specify the size of each message. Therefore, we exploit the higher-level abstraction provided by the Boost.MPI library (http://www.boost.org/). Coupled with the Boost Serialization library, it enables processes to exchange class instances, making the tasks of gathering, packing and unpacking the underlying data transparent. Given that OpenMP (http://openmp.org/) supports the C++ language, no adaptation or additional library are required.

[6]Our implementation applies, at a lower level, the bullx MPI library recommended by the Bull supercomputer manufacturer. Using the current version of this library (bullx MPI 1.2.8.4), MPI_THREAD_MULTIPLE is not supported.

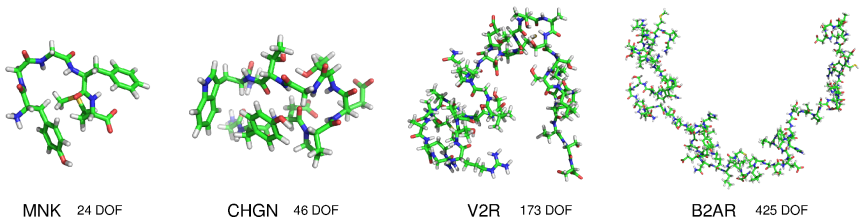

Figure 5: Models of the four molecules considered for the experiments.

5. Experiments

This section presents an empirical performance analysis of the proposed algorithm. First, we present the problems considered for this analysis, as well as the specifications of the computers we used. Then, the performance of the sequential algorithm, running on a single core, is analyzed to identify the most computationally expensive operations. The performance of the parallelized algorithm is then evaluated on a multi-core processor and on a cluster of processors, showing the interest of the hybrid approach.

5.1. Problem studied

We have evaluated the parallel MultiTRRT algorithm on several energy landscape exploration problems involving flexible biomolecules of different sizes. The number of degrees of freedom (DOF) of the molecule defines the dimensionality n of the space being explored. For most path planning algorithms, the theoretical time complexity grows exponentially with n. Although sampling-based algorithms are in practice less sensitive to dimensionality, computing time also increases significantly with n. We investigate two relatively small peptides, met-enkephalin [48] and chignolin [49], and the intrinsically disordered regions of the vasopressin 2 receptor [50] and a β-2 adrenergic receptor [51]. Figure 5 shows models of these four molecules. Hereafter, we will refer to these four molecules as MNK, CHGN, V2R and B2AR, respectively. The conformational exploration was performed using an internal-coordinate representation of the molecules, assuming constant bond lengths and bond angles. The number of

DOF for MNK, CHGN, V2R and B2AR are 24, 46, 173 and 425, respectively. For the four molecules, the energy landscape exploration was started from a set of m randomly sampled configurations. More precisely, we generated initial configurations using random sampling followed by local energy minimisation in order to obtain acceptable structures. In addition, a minimum distance was imposed between these initial configurations in order to maximize space coverage. The problem then consisted of building exploration trees rooted at these m initial states, and eventually to find paths connecting all of them. In the experiments presented below, we used $m = \{32, 90\}$. The same problem was solved for all instances tested, in which the number of threads and processors were varied.

5.2. Computer architecture

For the evaluation of the sequential algorithm and the multi-threaded implementation, we used a server with the following features: Intel® Core™ i7 processor at 2.8 GHz, 16 cores, 32 GB RAM.

The evaluation of the hybrid algorithm was performed on the EOS supercomputer at CALMIP (Centre de Calcul Midi-Pyrénées). EOS is a Bull supercomputer with 612 Intel® Ivy Bridge processors at 2.8 GHz, with 20 cores and 64 GB RAM per processor, and a InfiniBand interconnect with a full data rate of 6.89 GB/s for inter-processors communication. For our experiments, we used up to 90 processors (i.e. up to 1800 cores).

5.3. Analysis of the sequential algorithm

Figure 6 shows the percentage of time that the algorithm is expending in the most computationally-expensive operations. The computational requirements may change as a function of the search tree size, and thus, we evaluate these costs for various tree sizes. For each molecule, the plot shows the time decomposition with respect to the exploration tree size. The sum of `extend` and `findNearestNeighbor` methods consumes 99% of the computational time for all cases. Note that the energy computation is actually performed inside

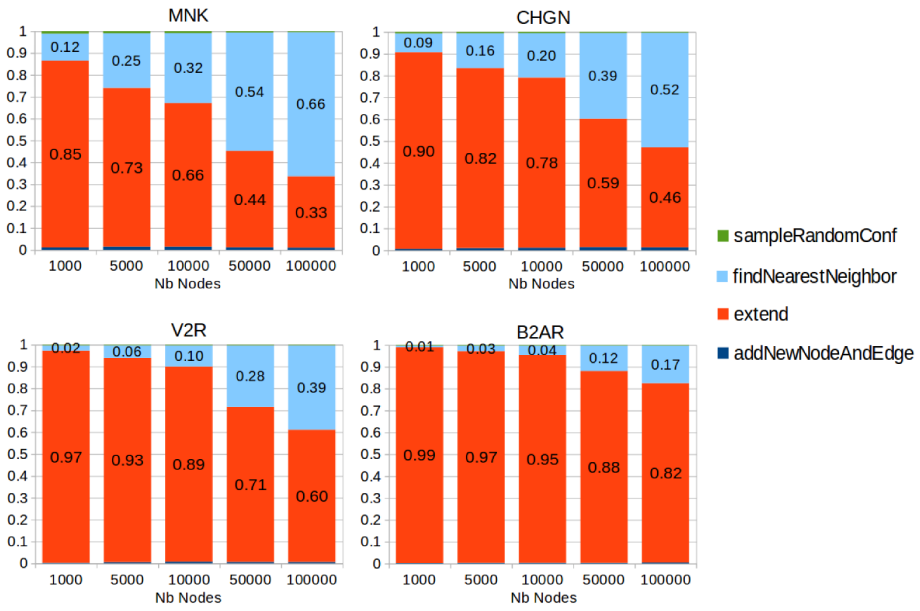

Figure 6: Computation time decomposition depending on the number of nodes in the trees for the four molecules under study.

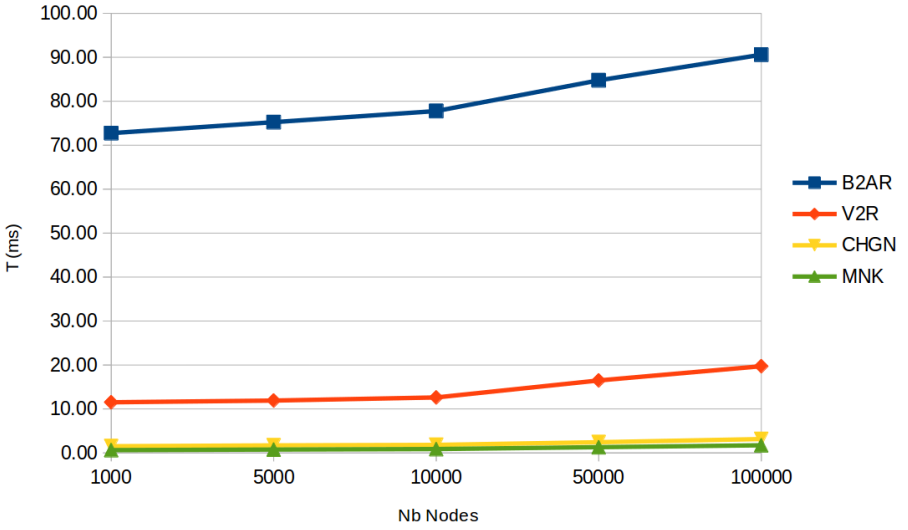

Figure 7: Evolution of the time per iteration with respect to the total number of nodes in the tree.

`extend`, and the value is stored for subsequent use. Therefore, the parallelization of these two operations is essential for performance improvements. It can be clearly seen in the figure that the time consumed by `extend` increases with the size of the molecule, being largely dominant for the two biggest molecules because of the computational cost of the energy computation. As expected, the time required by `findNearestNeighbor` becomes significant when the trees grow above several thousands of nodes. `sampleRandomConf` and `addNewNodeAndEdge` operations are very fast in all the cases, representing less than 0.1% of the total time. Therefore, introducing an OpenMP critical section to protect memory writing inside `addNewNodeAndEdge` does not represent significant overhead.

Figure 7 shows that the average time per iteration increases slowly with the number of nodes in the tree. The reason is that the most time-consuming operation, `extend`, which involves costly energy computation, is independent of the trees size. For this same reason, the time per iteration varies significantly with the size of the molecule, since cost of energy computation increases quadratically with the size of the protein. Compared to the smallest molecule, MNK, the time per iteration increases by one order of magnitude for V2R, and by almost 2 orders of magnitude for B2AR.

5.4. Analysis of the multi-threaded algorithm running on a single processor

As a preamble, before the implementation and analysis of the hybrid parallelization approach, we analyzed the performance of the Multi-TRRT algorithm running on a multi-core processor. The goal was to evaluate the potential interest of a larger-scale parallelization.

The usual metric to evaluate the performance of a parallel algorithm over its sequential counterpart is the speed-up $S(k) = t_s/t_p(k)$. Were t_s is the time needed to solve the problem working with one thread: $t_s = t_p(1)$, and $t_p(k)$ is the running time when k threads are used. The results presented in Figure 8 show that the performance strongly depends on the size of the molecule. The plot corresponds to CPU times averaged over 5 executions for each instance. For a small system such as MNK, a maximum speed-up of around 4.5 is reached

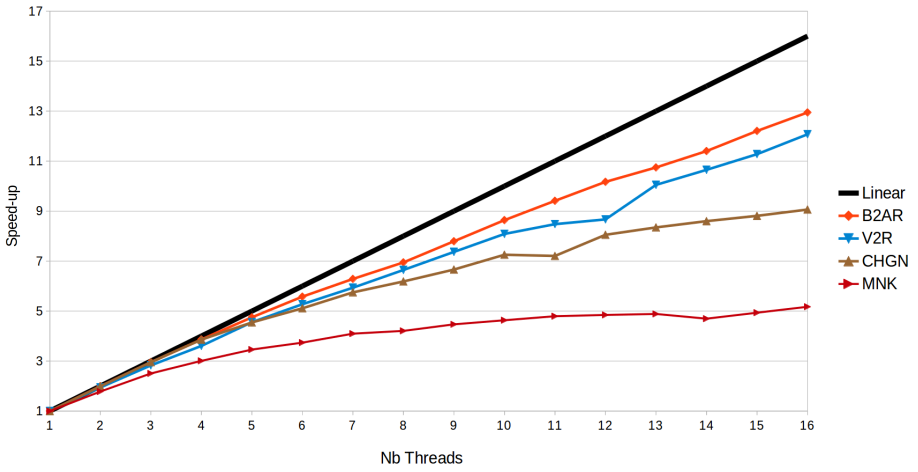

Figure 8: Evolution of the speed-up of the parallel algorithm running on a single (multi-core) processor for the four molecules. As a reference, the black line represent the linear speed-up.

for 11 threads. Then, the performance does not improve with a larger number of threads. For CHGN, the speed-up curve also tends to show an asymptotic shape, as for MNK, but with a maximum value approaching 8 for 16 threads. The reason of this limited performance gain is that the time per iteration of the main loop of the Multi-TRRT algorithm is very short for small systems (see Figure 7). Therefore, when the number of threads increases, the time spent waiting for access to critical sections starts to be significant, eventually becoming a bottleneck. Indeed, large-scale parallelization is useless for small molecules. For large systems such as V2R and B2AR, however, the speed-up increases almost linearly from 1 to 16 threads, with maximum speed-up values around 12 and 13, respectively. This shows that, in principle, better performance gains can be obtained with a larger number of threads.

In many cases, the speed-up is not the only criterion to assess the quality of a parallel algorithm. For instance, in our case, it is important to verify that the quality of the exploration and of the solution paths do not degrade with an increasing number of threads.

A side effect of the parallelization of RRT-based algorithms is that some

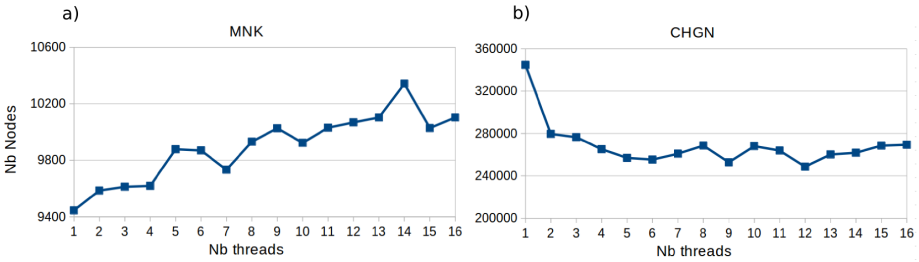

Figure 9: Evolution of the total number of nodes needed to solve the transition path finding problem depending on the number of threads for the two peptides: a) MNK, b) CHGN.

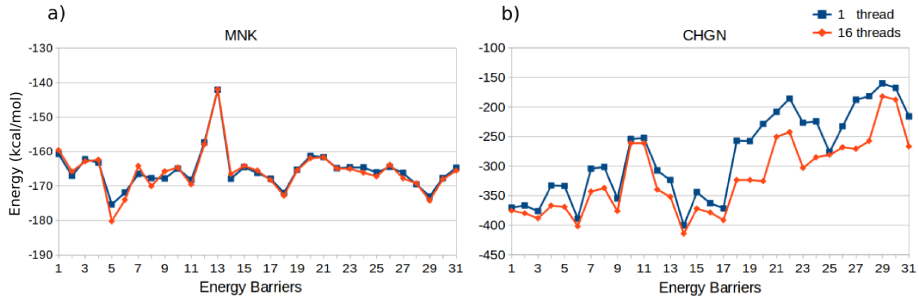

Figure 10: Energy profiles along the solution paths obtained using 1 thread and 16 threads for the two peptides: a) MNK, b) CHGN.

redundancy in the exploration can be introduced by multiple threads simultaneously creating nodes in nearby regions of the space. A simple test to detect if this happens is to look at the size of the trees (i.e. number of nodes) required to solve the same problem while varying the number of threads. An increasing number of nodes with the number of threads would mean a degradation of the exploration quality. As shown in Figure 9.a, the total number (averaged over 100 runs of the algorithm) of nodes in the trees required to find paths connecting 32 initial configurations for MNK increases slightly with the number of processors (up to 10%). This shows that, for problems in relatively low dimension, there is some redundancy in the exploration performed by the parallel version of Multi-TRRT. However, this undesired behaviour disappears when the dimension increases. This can be clearly seen in Figure 9.b for CHGN, which shows that the number of nodes in the trees required to solve the problem is almost

constant, independent of the number of threads. Surprisingly, in this case, the number of nodes is larger for a single thread (i.e. for the sequential algorithm), which actually shows that the parallel algorithm performs a more efficient exploration in high-dimensional spaces, due to what was called the "OR parallel effect" [34]. As each thread performs its own sampling of the space, when multiple threads are involved, the parallel algorithm reaches smaller-size solutions than the sequential one, on average. This phenomenon is more important in problems containing "narrow passages", corresponding to saddle regions in the energy landscape of the molecules, which require intensive sampling to be found.

To evaluate the quality of the paths obtained by the parallel implementation of Multi-TRRT compared to the sequential one, we can compare the corresponding energy profiles. It is worth recalling that the solution provided by the algorithm is a set of connected trees from which a path connecting the 32 initial configurations can be extracted. We can obtain a simplified representation of the energy profile by identifying the highest-energy configuration (i.e. the transition state) between each pair of initial configurations directly connected along the path. Figure 10.a shows that the energy profiles (averaged over 100 runs) of the solutions obtained with the sequential and the parallel versions (using 16 cores) of Multi-TRRT are very similar in the case of MNK, thus demonstrating that the quality of the solutions is preserved for problems in relatively low dimension. In higher dimension, as shown in Figure 10.b for CHGN, the quality of the paths are better for the solutions obtained by the parallel algorithm running with 16 threads compared to the sequential algorithm. This is also a consequence of the "OR parallel effect", which yields a better sampling of high-dimensional spaces with Multi-TRRT when several processes run in parallel.

5.5. Analysis of hybrid algorithm

We have evaluated the performance of the hybrid parallelization of Multi-TRRT for the two largest molecules: V2R and B2AR. For this experiment, instead of measuring the time required to connect the m initial configurations

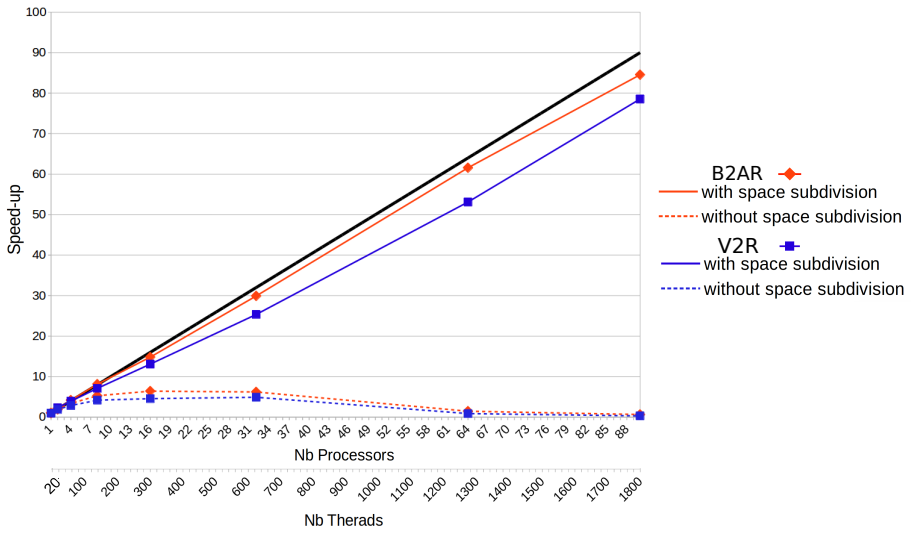

Figure 11: Evolution of the speed-up of the parallel algorithm with respect to the number of processors (working with 20 threads per processor) for the two largest molecules. Results are presented for Multi-TRRT with and without space subdivision. As a reference, the black line represents the linear speed-up.

(we used $m = 90$ for this experiment), we have measured the time required to generate 200,000 nodes. The reason is that the former involves a larger variance than the latter, therefore requiring a larger number of runs in order to obtain a meaningful average value. As we have seen in the previous subsection, the number of nodes needed to solve a problem is almost constant, independent of the number of threads/processes, and the variance is very low. Therefore, the time to generate a given number of nodes is a good indicator of the performance, and only requires a few runs per instance (10 in our case) to obtain statistically meaningful values.

Figure 11 shows the speed-up of the algorithm with respect to the number of processors. Experiments were performed for the implementation described in this paper and for a simpler version not including the adaptive space subdivision approach. With space subdivision, the speed-up increase is linear at the beginning, extending to 4 processors for V2R and up to 8 processors for B2AR. Then, the performance gains decrease slightly for both molecules. Nevertheless, the

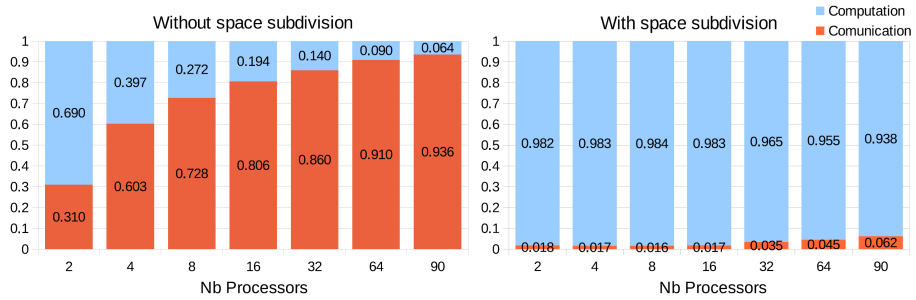

Figure 12: Proportion of the time used for computation and for communication depending on the number of processors without space subdivision (left) and with space subdivision (right).

speed-up using 90 processors is 85 for the largest molecule, B2AR. This means a parallel efficiency (i.e. speed-up per processor) around 0.94, being very close to the ideal value. Without the space subdivision approach, the performance of the algorithm is drastically different. In this case, each processor needs to communicate with all the others each time a new node is created, and nearest neighbor searches have to be performed for all the trees. This causes a very rapid collapse of the overall exploration process, even when a small number of processors are used. For both molecules, the speed-up reaches a plateau around a value of 5-6, extending from 8 processors up to 32 processors. Then, the speed-up starts decreasing, reaching a value below 1 for 90 processors. In summary, these results shows that the proposed parallelization strategy is very efficient, and that the space subdivision approach to reduce inter-processor communication is very effective.

We have made additional tests and measurements to evaluate the performance of the algorithm more accurately. Figure 12 showcases the proportion of the time used for computation and for communication, depending on the number of processors, for the conformational exploration of B2AR. Using the space subdivision approach, the percentage of the time required for communication is very low, even when using 90 processors (around 6% in the worst case). On the contrary, communication cost rapidly becomes a bottleneck without space subdivision. This explains the very poor performance of this simpler version of

Table 1: Time per processor (in seconds) for a single run of the parallel Multi-TRRT.

Nb Proc.	**Task**	**Min.**	**Max.**	**Average**	**Std. dev.**
2	*Computation*	1685.5043	1685.5326	1685.5184	0.0001
	Communication	6.5855	10.1001	8.3428	0.2106
4	*Computation*	822.6771	823.8928	823.1394	0.0006
	Communication	5.2486	6.6333	5.6552	0.1014
8	*Computation*	421.0210	423.6831	421.7080	0.0019
	Communication	2.8500	5.1652	3.2154	0.2300
16	*Computation*	206.2155	213.0075	208.0654	0.0075
	Communication	1.5405	4.4856	0.5990	0.3628
32	*Computation*	113.9660	121.0360	118.2404	0.0124
	Communication	1.0749	5.1344	1.3101	0.5254
64	*Computation*	63.9305	69.9495	65.9272	0.0172
	Communication	0.9574	5.0125	1.4440	0.4038
90	*Computation*	33.0473	33.0794	33.0572	0.0002
	Communication	1.6552	6.1581	2.6923	0.1772

the parallel Multi-TRRT algorithm.

Table 1 presents statistics about the time required by each processor running the parallel Multi-TRRT algorithms to generate its corresponding part of the 200,000 configurations for B2AR. The results correspond to a single run of the algorithm (i.e. they are not averaged over several runs) in order to avoid a possible smoothing of the results regarding load balance, which is represented here by the standard deviation. One can observe that computation time is very similar for all the processors. Communication time is more variable. The reason is that some processors construct exploration trees having more connections to other trees, thus requiring more communication compared to processors that construct trees in more isolated areas. Note however that communication time is very small compared to computation time in all the cases. Thus, these results show a very good load balance among the processors. Inside each processor, the load of all the cores, which collaboratively construct one or several trees, is almost identical.

In the proposed implementation, one of the processors (called master) exe-

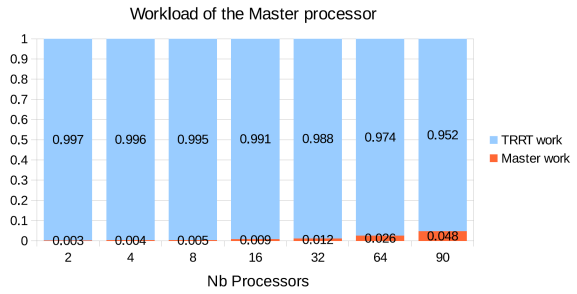

Figure 13: Proportion of the time spent by the master processor executing coordination tasks (Master work) and the exploration task (TRRT work).

cutes a few specific operations related to the connection of different exploration trees, in addition to the execution of the TRRT algorithm as the other processors. Figure 13 represents the proportion of the time required by each of these tasks, depending on the number of processors, for the conformational exploration of B2AR. Since the coordination task involves simple and computationally cheap operations, the percentage of the time required to execute this task is very low. Even in the worst case, for 90 processors, the coordination task (including communication) requires only about 5% of the capacity of the master processor. In conclusion, the master processors is far from being a bottleneck.

6. Conclusions

Nowadays, most HPC systems are clusters of multi-core processors. We have presented a parallel implementation of Multi-TRRT to efficiently exploit these type of architectures, combining distributed-memory parallelization using MPI with shared memory-parallelization using OpenMP. Such a hybrid parallelization strategy clearly outperforms our previous fully-distributed implementations of RRT-like algorithms [34], significantly reducing communication overhead and memory requirements. This implementation is also very flexible, since the algorithms can be run on a single multi-core processor (without communication requirements) or on a large computer cluster without any modification in the code. The adaptive space subdivision approach is a key component of the pro-

posed parallelization strategy. It drastically reduces the computational cost associated with inter-processor communication and nearest neighbor search.

Although the paper is focused on the application of the proposed parallel algorithm to highly-flexible biomolecules, the explanations concerning the methods can be easily extracted from the application context. The principle is very general, and could be applied to other sampling-based path search/planning algorithms applied in different domains. Indeed, we expect that our work will be a source of inspiration for the parallelization of related methods.

The analysis presented in this paper shows that the performance gain provided by the parallel algorithm depends on the complexity of the problem (i.e. on the size of the molecule). For relatively small systems such as peptides of up to 10-15 residues, using multiple threads in a single computer to collaboratively construct the exploration trees is probably the best choice. In this case, the performance gain provided by a larger computer cluster is not very significant, presenting an asymptotic profile. However, for larger systems such as IDPs, the speed-up obtained by the hybrid approach increases almost linearly with the number of processors, even when this number is large. Indeed, these encouraging results demonstrate the applicability of the parallelized Multi-TRRT algorithm to characterize the conformational landscape of large disordered proteins. We expect that the methodological advances presented in this paper will contribute in the near future to a better understanding of such challenging biomolecules, which are involved in key biological processes and neurodegenerative pathologies.

7. Acknowledgements

The authors would like to thank Leonard Marsault for his help with the integration of FLANN, as well as Nicolas Renon and Mickaël Duval for their technical support for the implementation and evaluation of the parallel algorithm. This research has been partially funded by the European Research Council under the European Union's H2020 Framework Programme (2014-2020) / ERC

Grant agreement number 648030 and by the French National Research Agency (ANR) under project ProtiCAD (project number: ANR-12-MONU-0015). Some of the experiments in this work were performed using the HPC resources of the CALMIP supercomputing center under the allocation 2016-P16032.

References

[1] D. Frenkel, B. Smit, Understanding Molecular Simulations: From Algorithms to Applications, 2nd Edition, Vol. 1 of Computational Science Series, Academic Press, 2001.

[2] D. J. Wales, Energy Landscapes: Applications to Clusters, Biomolecules and Glasses, Cambridge University Press, 2003.

[3] M. Karplus, J. A. McCammon, Molecular dynamics simulations of biomolecules, Nature Structural & Molecular Biology 9 (2002) 646–652.

[4] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E.Teller, Equation of state calculations by fast computing machines, Journal of Chemical Physics 21 (1953) 1087–1092.

[5] I. Al-Bluwi, T. Siméon, J. Cortés, Motion planning algorithms for molecular simulations: A survey, Computer Science Review 6 (4) (2012) 125–143.

[6] B. Gipson, D. Hsu, L. E. Kavraki, J.-C. Latombe, Computational models of protein kinematics and dynamics: Beyond simulation, Annual Review of Analytical Chemistry 5 (2012) 273–291.

[7] A. Shehu, Probabilistic search and optimization for protein energy landscapes, in: S. Aluru, M. Singh (Eds.), Handbook of Computational Molecular Biology, 2nd Edition, Computer & Information Science Series, Chapman & Hall/CRC, 2013, in press.

[8] L. Jaillet, F. J. Corcho, J.-J. Pérez, J. Cortés, Randomized tree construction algorithm to explore energy landscapes, Journal of Computational Chemistry 32 (16) (2011) 3464–3474.

[9] D. Devaurs, T. Siméon, J. Cortés, A multi-tree extension of the Transition-based RRT: Application to ordering-and-pathfinding problems in continuous cost spaces, in: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014.

[10] D. Devaurs, K. Molloy, M. Vaisset, A. Shehu, T. Siméon, J. Cortés, Characterizing Energy Landscapes of Peptides using a Combination of Stochastic Algorithms, IEEE Transactions on NanoBioscience 14 (5) (2015) 545–552.

[11] S. M. LaValle, Rapidly-exploring Random Trees: a new tool for path planning, Tech. Rep. TR: 98-11, Computer Science Dept., Iowa State University (1998).

[12] T. N. Cordeiro, F. Herranz-Trillo, A. Urbanek, A. Estaña, J. Cortés, N. Sibille, P. Bernadó, Small-angle scattering studies of intrinsically disordered proteins and their complexes, Current Opinion in Structural Biology 42 (Supplement C) (2017) 15 – 23.

[13] V. N. Uversky, C. J. Oldfield, A. K. Dunker, Intrinsically disordered proteins in human diseases: Introducing the D2 concept, Annual Review of Biophysics 37 (1) (2008) 215–246.

[14] M. M. Babu, R. van der Lee, N. S. de Groot, J. Gsponer, Intrinsically disordered proteins: regulation and disease, Current Opinion in Structural Biology 21 (3) (2011) 432 – 440.

[15] H. L. Nguyen, H. Khanmohammadbaigi, E. Clementi, A parallel molecular dynamics strategy, Journal of Computational Chemistry 6 (1985) 634.

[16] J. Li, Z. Zhou, R. J. Sadus, Parallelization Algorithms for Three-Body Interactions in Molecular Dynamics Simulation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 374–382.

[17] A. Heinecke, W. Eckhardt, M. Horsch, H.-J. Bungartz, Parallelization of MD Algorithms and Load Balancing, Springer International Publishing, Cham, 2015, pp. 31–44.

[18] E. Lindahl, B. Hess, D. van der Spoel, Gromacs 3.0: a package for molecular simulation and trajectory analysis, Molecular modeling annual 7 (8) (2001) 306–317.

[19] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, K. Schulten, Scalable molecular dynamics with namd, Journal of Computational Chemistry 26 (16) (2005) 1781–1802.

[20] Y. Sugita, Y. Okamoto, Replica-exchange molecular dynamics method for protein folding, Chemical Physics Letter 314 (1999) 141–151.

[21] F. Rao, A. Caflisch, Replica exchange molecular dynamics simulations of reversible folding, Journal of Chemical Physics 119 (2003) 4035–4042.

[22] J. S. Rosenthal, Parallel computing and monte carlo algorithms, Far East Journal of Theoretical Statistics 4 (2000) 207–236.

[23] I. Strid, Efficient parallelisation of Metropolis-Hastings algorithms using a prefetching approach, Computational Statistics & Data Analysis 54 (11) (2010) 2814 – 2835.

[24] D. Gront, A. Kolinski, Efficient scheme for optimization of parallel tempering Monte Carlo method, Journal of Physics: Condensed Matter 19 (2007) 036225.

[25] R. H. Swendsen, J.-S. Wang, Replica monte carlo simulation of spin-glasses, Physical Review Letters 57 (1986) 2607–2609.

[26] J.-C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, 1991.

[27] S. M. LaValle, Planning Algorithms, Cambridge University Press, Cambridge, U.K., 2006.

[28] D. Henrich, Fast motion planning by parallel processing – a review, Journal of Intelligent and Robotic Systems 20 (1) (1997) 45–69.

[29] J. Ichnowski, R. Alterovitz, Scalable multicore motion planning using lock-free concurrency, IEEE Transactions on Robotics 30 (5) (2014) 1123–1136.

[30] J. Bialkowski, S. Karaman, E. Frazzoli, Massively parallelizing the RRT and the RRT*, in: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011, pp. 3513–3518.

[31] J. Pan, D. Manocha, Gpu-based parallel collision detection for fast motion planning, International Journal of Robotics Research 31 (2) (2012) 187–200.

[32] N. M. Amato, L. K. Dale, Probabilistic roadmap methods are embarrassingly parallel, in: Proc. IEEE International Conference on Robotics and Automation (ICRA), 1999, pp. 688–694.

[33] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, L. E. Kavraki, Sampling-based roadmap of trees for parallel motion planning, IEEE Transactions on Robotics 21 (4) (2005) 597–608.

[34] D. Devaurs, T. Siméon, J. Cortés, Parallelizing RRT on large-scale distributed-memory architectures, IEEE Transactions on Robotics 29 (2) (2013) 571–579.

[35] L. E. Kavraki, P. Švestka, J.-C. Latombe, M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Transactions on Robotics and Automation 12 (4) (1996) 566–580.

[36] D. Challou, D. Boley, M. Gini, V. Kumar, C. Olson, Parallel search algorithms for robot motion planning, in: K. Gupta, A. P. del Pobil (Eds.), Practical Motion Planning in Robotics: Current Approaches and Future Directions, John Wiley & Sons Ltd., 1998, pp. 115–131.

[37] S. Caselli, M. Reggiani, ERPP: An experience-based randomized path planner, in: Proc. IEEE International Conference on Robotics and Automation (ICRA), 2000, pp. 1002–1008.

[38] S. Carpin, E. Pagello, On parallel RRTs for multi-robot systems, in: Proc. International Conference of the Italian Association for Artificial Intelligence (AI*IA), 2002, pp. 834–841.

[39] I. Aguinaga, D. Borro, L. Matey, Parallel RRT-based path planning for selective disassembly planning, The International Journal of Advanced Manufacturing Technology 36 (11-12) (2008) 1221–1233.

[40] S. A. Jacobs, N. Stradford, C. Rodriguez, S. Thomas, N. M. Amato, A scalable distributed rrt for motion planning, in: Proc. IEEE International Conference on Robotics and Automation (ICRA), 2013, pp. 5088–5095.

[41] S. M. LaValle, J. J. Kuffner, Rapidly-exploring random trees: progress and prospects, in: B. R. Donald, K. M. Lynch, D. Rus (Eds.), Algorithmic and Computational Robotics: New Directions, A. K. Peters, Wellesley, MA, 2001, pp. 293–308.

[42] D. Hsu, R. Kindel, J.-C. Latombe, S. M. Rock, Randomized kinodynamic motion planning with moving obstacles, The International Journal of Robotics Research 21 (3) (2002) 233–255.

[43] M. Otte, N. Correll, C-FOREST: Parallel shortest-path planning with super linear speedup, IEEE Transactions on Robotics 29 (2013) 798–806.

[44] L. Jaillet, J. Cortés, T. Siméon, Sampling-based path planning on configuration-space costmaps, IEEE Transactions on Robotics 26 (4) (2010) 635–646.

[45] P. Kollman, R. Dixon, W. Cornell, T. Fox, C. Chipot, A. Pohorille, The development/application of a "minimalist" organic/biochemical molecular mechanic force field using a combination of ab initio calculations and experimental data, in: W. van Gunsteren, P. Weiner, A. Wilkinson (Eds.), Computer Simulation of Biomolecular Systems, Vol. 3 of Computer Simulations of Biomolecular Systems, Springer Netherlands, 1997, pp. 83–96.

[46] D. Devaurs, T. Siméon, J. Cortés, Enhancing the Transition-based RRT to deal with complex cost spaces, in: Proc. IEEE International Conference on Robotics and Automation (ICRA), 2013, pp. 4105–4110.

[47] M. Muja, D. G. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: In VISAPP International Conference on Computer Vision Theory and Applications, 2009, pp. 331–340.

[48] G. H. Paine, H. A. Scheraga, Prediction of the native conformation of a polypeptide by a statistical-mechanical procedure. III. Probable and average conformations of enkephalin, Biopolymers 26 (7) (1987) 1125–1162.

[49] D. Satoh, K. Shimizu, S. Nakamura, T. Terada, Folding free-energy landscape of a 10-residue mini-protein, chignolin, FEBS Letters 580 (14).

[50] A. K. Shukla, G. H. Westfield, K. Xiao, et al., Visualization of arrestin recruitment by a G protein-coupled receptor, Nature 512 (2014) 218–222.

[51] S. Granier, S. Kim, A. M. Shafer, V. R. Ratnala, J. J. Fung, R. Zare, B. Kobilka, Structure and conformational changes in the c-terminal domain of the beta2-adrenoceptor: insights from fluorescence resonance energy transfer studies, Journal of Biological Chemistry 282 (2007) 13895–13905.