A Scalable Method for Parallelizing Sampling-Based Motion Planning Algorithms

Sam Ade Jacobs, Kasra Manavi, Juan Burgos, Jory Denny, Shawna Thomas and Nancy M. Amato

Abstract— **This paper describes a scalable method for parallelizing sampling-based motion planning algorithms. It subdivides configuration space (C-space) into (possibly overlapping) regions and independently, in parallel, uses standard (sequential) sampling-based planners to construct roadmaps in each region. Next, in parallel, regional roadmaps in adjacent regions are connected to form a global roadmap. By subdividing the space and restricting the locality of connection attempts, we reduce the work and inter-processor communication associated with nearest neighbor calculation, a critical bottleneck for scalability in existing parallel motion planning methods.**

We show that our method is general enough to handle a variety of planning schemes, including the widely used Probabilistic Roadmap (PRM) and Rapidly-exploring Random Trees (RRT) algorithms. We compare our approach to two other existing parallel algorithms and demonstrate that our approach achieves better and more scalable performance. Our approach achieves almost linear scalability on a 2400 core LINUX cluster and on a 153,216 core Cray XE6 petascale machine.

I. Introduction

Motion planning is the problem of finding a valid path (e.g., collision-free) for a robot (or other movable object) from a specified start configuration to a goal configuration in an environment [11]. Motion planning plays a significant role in a number of application areas outside robotics including computer animation [17], [3], virtual prototyping [10], [4], and computational biology [25], [5], [26].

Sampling-based motion planning algorithms have been highly successful at solving previously unsolved problems [11], and much research has focused on developing more sophisticated variants of them. Sampling-based motion planning algorithms are broadly classified as either tree-based or graph-based. The tree-based approach has many variants but the basic idea is to grow one or more trees starting from a given (start) configuration in the robot's configuration space (C-space). The graph-based approach is well suited for multi-query use and is generally performed in two phases: a preprocessing phase, during which a graph representing valid paths is constructed, and a query phase, during which the start and goal configurations are connected to the precomputed graph. A simple graph search is used to extract a path. Sampling-based approaches are efficient and can be applied to high dimensional problems. While not guaranteed to find a solution, they are probabilistically complete, meaning that the probability of finding a solution given one exists increases with the number of samples generated [16].

Despite their advantages, the efficiency of sampling-based motion planning algorithms degrades as the ratio of obstacle space to free space increases – common in high dimensional problems. Thus, substantial resources in time and hardware are still required to solve computationally intensive applications. For example, the authors in [28] reported that it took several hours on a desktop PC to compute a roadmap modeling the folding motion of a small protein using coarse approximations for energy calculations. This time increases to several weeks if more accurate energy calculations are used or if a larger protein were studied.

One solution to this resource problem may lie in parallel computing. For many application areas, parallel processing offers the advantage of not only reducing computation time, but also improving the solution quality and enabling larger problems to be solved than were feasible before.

In this work, we present a strategy for parallelizing sampling-based motion planning algorithms. Our strategy uses C-space subdivision to achieve scalability. First, the planning space is separated into (possibly overlapping) regions, at least one per processor. Then, each processor independently applies a sampling-based planner in its region and produces a regional roadmap. Finally, adjacent regional roadmaps are connected to form a global roadmap. By subdividing the space, we reduce the amount of work and inter-processor communication required for nearest neighbor calculations, which has been seen to be a critical bottleneck for scalability in previous methods. Our approach was implemented using the STAPL (Standard Template Adaptive Parallel Library) framework for parallel C++ code, a research project in the Parasol Lab at Texas A&M University [8], [27].

Contribution. Key contributions of this work include:

- The first reported work in parallel sampling-based motion planning based on C-space subdivision.
- Experiments that show we achieve better and more scalable performance on thousands of processors than two previous parallel sampling-based planners.
- An approach that is compatible with any sampling-based algorithm, including the Probabilistic Roadmap (PRM) and Rapidly-exploring Random Trees (RRT) algorithms.

This research supported in part by NSF awards: CRI-0551685, CCF-0833199, CCF-0830753, IIS-096053, IIS-0917266, NSF/DNDO award 2008-DN-077-ARI018-02, by the DOE NNSA under the Predictive Science Academic Alliances Program grant DE-FC52-08NA28616, by THECB NHARP award 000512-0097-2009, by Chevron, IBM, Intel, Oracle/Sun and by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST). This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

The authors are with the Parasol Lab., Dept. of Computer Science and Engineering, Texas A&M Univ., College Station, Texas, 77843-3112, USA. {sjacobs,kmmanavi,jlb4248,jdenny,sthomas,amato}@cse.tamu.edu

II. Related Work

For years, researchers have proposed and studied different types of parallel algorithms for motion planning. For a detailed survey of early work in general parallel motion planning, see [13]. Recently, research efforts have focused on parallel sampling-based motion planning due to the success of sequential sampling-based motion planning in solving high dimensional problems [14]. We first review sequential sampling-based motion planning approaches, followed by their parallel counterparts, and conclude with related work on C-space subdivision.

A. Sampling-based Motion Planning (SBMP)

One well established sampling-based motion planning approach is the Probabilistic Roadmap (PRM) [16]. PRMs first construct a graph $G = (V, E)$, called a roadmap, to capture the connectivity of C-space. A node in the graph represents a valid (e.g., collision-free) placement of the robot (movable object), and an edge is added between two nodes if a simple path can be defined and validated by a local planner. In the initial method, nodes are generated using uniform random sampling and connections are attempted between a node and its k-nearest neighbors as computed using some distance metric (e.g., Euclidean). Once the roadmap is constructed, query processing is done by connecting the start and goal configurations to the roadmap and extracting a path through the roadmap between them. Many variants of PRMs have been proposed that bias node generation or connection in various ways [11].

Tree-based methods, such as the Rapidly-exploring Random Tree (RRT) [19] and Hsu's expansive planner [15], are other popular approaches to sampling-based motion planning. RRTs attempt to grow a tree from the start configuration expanding outward into the unexplored areas of C-space. They first generate a uniform random sample and identify the nearest node in the tree to that sample. They then attempt to make a step from the nearest node toward the generated sample and add the step to the tree if successful. When solving a query, RRTs continue until the goal configuration can be connected to the tree in a similar fashion. RRT-connect [18] is a variant of the original RRT algorithm that grows two trees, one from the start configuration and one from the goal configuration, until the two trees can be connected.

B. Parallel Sampling-based Motion Planning

The sequential PRM algorithm was first parallelized in [2] and then specifically for protein folding applications in [28]. Both papers present a similar parallel approach. Each processor generates an "equal" number of nodes in the entire C-space in parallel and adds them to the roadmap. Then, each processor attempts to connect its nodes with their k-nearest neighbors in the entire roadmap. The major drawback of this approach was the all-to-all computation and communication involved in the $O(n^2)$ method used to find nearest neighbors which did not scale to large systems or problem sizes.

A basic parallelization of RRTs is given in [9] where the problem is replicated on each processor. Each processor then constructs its own RRT and concurrently explores the entire C-space along with all the other processors. The first processor to find a solution sends a termination message to other processors. More recent research on parallelization of RRTs is presented in [12], [6]. While the work in [12] extends the initial idea of RRT parallelization, the work in [6] explores GPU implementation of parallel RRT and RRT* algorithms with a primary focus on parallelizing the collision detection phase.

Parallel Sampling-based Roadmap of Trees (pSRT) [1], [22], [23] combines the multiple query sampling characteristics of PRMs with the efficient local planning capabilities of single query of RRTs. Unlike previous approaches, the nodes of a pSRT roadmap are trees instead of individual configurations. The collections of these trees form the roadmap. Connections between trees are attempted between closest pairs of configurations between the two trees. The authors adopted the scheduler (master) – processor (slave) architecture. Each slave processor computes a predefined number of trees in the entire C-space. The master is responsible for arbitration of tree ownership, nearest neighbor computations, and determination of which pairs of trees to attempt for connection. Edge validation is distributed to the slave processes.

In most existing work surveyed, we identify inter-processor communication, redundant computation, and load imbalance (in master-slave architecture) as the key bottlenecks to scalable performance. To address some of these drawbacks, we propose a C-space subdivision approach.

C. C-space Subdivision

The concept of C-space subdivision has been proposed and used in many existing sequential motion planning algorithms. One of the earliest complete (or exact) motion planning algorithms computes an exact representation of C-space by uniformly dividing it along the robot's degrees of freedom into cells [7]. However, this approach is not practical for high dimensional problems.

Feature sensitive motion planning [20], [21] proposes a supervised method of recursively breaking up an environment into regions and classifying these regions as free, clutter, narrow, or blocked by comparing region features to a database of known region types. Roadmaps are constructed in each region and recombined to form a final roadmap.

RESAMPL [24] subdivides the C-space into local regions based on an initial sampling of the entire space. As a partitioning strategy, RESAMPL first generates a small set of samples, both valid and invalid, in the entire space. Some of these samples, selected from the set randomly, become representative samples for the local regions. Region sizes are determined by the distance of the representative sample to its k-nearest neighbors in the initial set.

Another space subdivision approach is the Approximate Cell Decomposition (ACD) method [29]. ACD subdivides the C-space into rectangular cells. Each generated cell is

labelled as $empty$ if it lies completely in free space, $full$ if it lies completely in obstacle space, or $mixed$ otherwise. PRM is combined with ACD to compute localized roadmaps by generating samples within these cells. The connectivity graph for adjacent cells in ACD is augmented with pseudo-free edges that are computed based on localized roadmaps.

The work in [20], [21] is the most closely related work to our own. The authors in [20], [21] present a C-space subdivision that uses a random splitting point from randomly selected positional degree of freedom (DOF) so as to define an orthogonal boundary to the selected DOF. This splitting process is repeated recursively until homogeneous but overlapping sets of regions are obtained. Homogeneity is defined according to a set of features measured for each region. Unlike the work in [20], [21], we do not focus on region identification. Rather, in this initial work, the regions were constructed in a regular manner that facilitated parallelization.

III. Strategy For Parallelizing Sampling-Based Motion Planning

A. Overview of Approach

An overview of our approach is given in Algorithm 1. Initially, the environment describing the obstacles and robot is subdivided into regions. A simple illustration of a 2D environment subdivided into four regions is shown in Figure 1(a). The subdivision is represented by a *region graph*, whose vertices represent regions and whose edges encode the adjacency information between regions. Figure 1(b) shows the region graph corresponding to the subdivision shown in Figure 1(a). The circles and straight lines connecting adjacent circles represent vertices and edges of the region graph, respectively.

Algorithm 1 Parallel Sampling-based Motion Planning

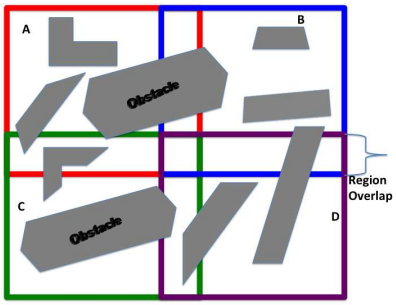
Input: An environment E, A set of motion planners S, number of regions N_R

Output: A roadmap graph G

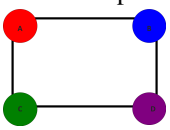
1: Decompose E into N regions
2: Make a region graph $R = (V_R, E_R)$ with V_R and E_R representing each region and adjacency information between regions, respectively
3: Independently and in parallel, construct roadmaps in each region using any desired planner $s \in S$
4: Connect regional roadmaps in adjacent regions to form a roadmap G for the entire problem

In Step 2, roadmaps are constructed independently, in parallel, in each region. Hence, this approach does not depend on the underlying sampling-based motion planning algorithm or strategy and can handle a variety of planning schemes.

In Step 3, we connect nearby regional roadmaps to form a roadmap representing the entire C-space. The region graph is the enabling infrastructure facilitating the process of connecting the region roadmaps. The region graph infrastructure aids identification of adjacent regions between which connections are attempted. In this way, communication is only limited to adjacent regions.



(a) A 2D environment subdivided into 4 regions with user-defined overlap between regions



(b) Region Graph

Fig. 1. Space subdivision

In the following, we describe each step in more detail.

B. Space Subdivision and Region Graph Construction

We subdivide a given environment by breaking up the planning space into a set of regions. While the approach is general, in this work, we consider the x, y, and z dimensions of the original workspace only. Each processor is then assigned at least one region and the task of building a regional roadmap in its assigned region(s). In this initial work, we maintain some user-defined overlap between regions to allow sampling in the portions of the space that are at the boundaries that may facilitate connection between regional roadmaps.

We made use of two different graphs as underlying data structures: the roadmap graph and the region graph.

The roadmap graph stores the nodes and edges representing the configurations sampled and the connections between the samples, respectively.

The region graph, shown in Figure 1(b), whose construction is described in Algorithm 2, represents the regions and the adjacencies between regions with its vertices and edges, respectively. This information is used to limit communication to adjacent regions. In addition, the region graph also maintains additional information that keeps track of the connected components in each region which is used when connecting adjacent regions.

C. Constructing Regional Roadmaps

Step 2 of Algorithm 1 involves construction of regional roadmaps. At this step, any of the existing sampling-based motion planning algorithms, such as PRM (and its variants) or RRT (and its variants) can be used. This step is independent of the sampling strategy employed. At this step, each processor independently generates and connects samples in

Algorithm 2 Region Graph Construction

Input: An environment E and the number of regions N_R.
Output: A region graph R.
Let $R = \emptyset$.
Let R_d = SubDivideSpace(E, N_R).
Add a vertex for each region r of R_d to R.
for all neighboring regions (r_1, r_2) ϵ R_d) **par do**
  Add the edge (r_1, r_2) to R.
end for
Return R.

its assigned region with no or minimal communication with other regions. The roadmaps built at this step are added to the roadmap graph. To facilitate and streamline the connection at the next step, we keep track of the size and a vertex representative for each connected component in the regional roadmap.

Algorithm 3 Region Roadmap Connection

Input: A region graph R, connection method, k number of candidates, local planner lp
Output: A roadmap graph G.
for all edges E ϵ R **par do**
  if (connection method == closest) **then**
    sourceCC = select k center of mass based closest CC to target region from $E.source$
    targetCC = select k center of mass based closest CC to source region from $E.target$
  end if
  if (connection method == largest) **then**
    sourceCC = select k largest CCs from $E.source$
    targetCC = select k largest CCs from $E.target$
  end if
  for all pairs($sourceCC, targetCC$) **do**
    if lp.IsConnectable($sourceCC, targetCC$) **then**
      Add the edge($sourceCC, targetCC$) to G.
    end if
  end for
end for
Return G.

D. Connecting Regional Roadmaps

The final step in constructing the full roadmap is to connect the regional roadmaps. Prior to this step, we track the sizes and number of connected components in each region. The regional graph stores this information which is input to the region connection algorithm shown in Algorithm 3. Other inputs to the algorithm include: k, the number of connections to be attempted between adjacent regions, the type of connection method, and a local planner used to verify connections.

For every edge identifying neighboring regions in the region graph, we attempt a connection between candidate node(s) of connected components in the source region to candidate node(s) of connected components in the target region. Even though our implementation is independent of which region connection method is used, in this work, we attempt to connect regions based on the size of connected components in each region and the distance between connected components across regions. For the size-based connection, we attempt connections between a user-defined k largest connected components from the source region and k largest connected components from the target region. For the distance-based connection, we attempt to connect the k-closest connected components between the regions based on the distance between them. This distance is computed between the centers of mass (a measure of average of all configurations in the connected component) of the two connected components.

IV. Analysis of the Algorithm

The original PRM algorithm as reported in [16] requires $O(N^2)$ time and $O(N)$ space to construct a roadmap with N configurations. This serves as the basis for our analysis and is used for the complexity of constructing a regional roadmap.

The overall time complexity of our approach as described in Algorithm 1 can be given as:

$$T = T_d(Env, n_r) + T_r(i)|V_R| + T_c(i,j)|E_R|$$

where T is the sum of the cost of space decomposition T_d for a given environment Env subdivided into n_r regions, the cost $T_r(i)$ of roadmap construction in region r_i, for all v_i ϵ V_R, and the cost $T_c(i,j)$ of connecting regional roadmaps in regions r_i and r_j, for all (r_i, r_j) ϵ E_R. This formulation makes the simplifying assumption that the cost of constructing regional roadmaps is the same for all regions and similarly for the cost of connecting regional roadmaps. If this is not the case, then the costs of these should be determined separately and summed.

Step 1 involves space decomposition. We assume p processors/tasks and that the regions are divided equally among the p processors. In this case, the regular workspace-based decomposition of C-space used in this paper, and its corresponding region graph, can be constructed in time $O(|V_R| + |E_R|)/p$ where V_R and E_R are the vertices and edges of the region graph, respectively.

Step 2 of Algorithm 1 involves the construction of the regional roadmaps. Since we are assuming there are p regional roadmaps, each of the same size, this implies they will have N/p nodes each, and hence the cost of (sequentially) constructing the PRM roadmap for each region will be $O((N/p)^2)$. If regional roadmaps are RRTs instead, one would use the cost of constructing an RRT of size N/p here instead, and similarly for any other desired approach for constructing a regional roadmap.

Most inter-processor communication occurs when connecting regional roadmaps. The region graph infrastructure helps to limit both computation and communication to adjacent regions. If we assume a naive connection attempt between every configuration in a region to every configuration in neighboring region, this worst case scenario will

result in $O((N/p)^2)$ edge computations plus the cost of communication between neighboring processors.

Thus, in summary, the time, work and space complexity of this approach can be given as $O((N/p)^2)$, $O((N^2)/p)$ and $O(N)$ respectively.

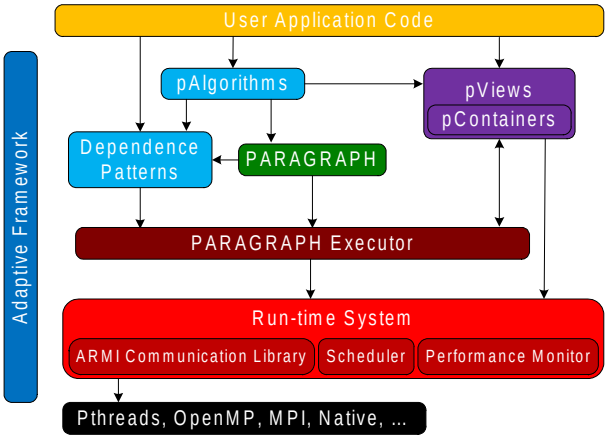
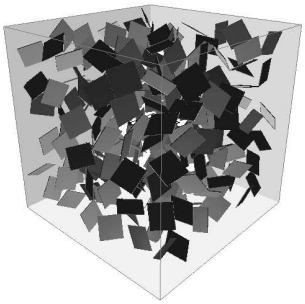
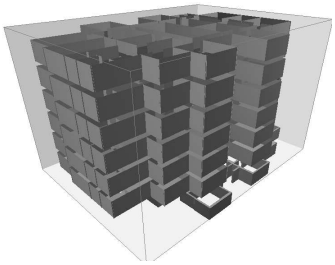


Fig. 2. STAPL software architecture.



(a) Clutter



(b) Building

Fig. 3. Environments studied

V. Implementation using STAPL

Our code was written in C++ using the Standard Template Library (STL) and the Standard Template Adaptive Parallel Library (STAPL) as supporting libraries. STAPL [8] is a platform independent superset of STL being developed in our lab. It provides a collection of building blocks for writing parallel programs. These building blocks (as shown in Figure 2) include a collection of parallel algorithms ($pAlgorithms$), parallel and distributed containers ($pContainers$), a general mechanism to access the data of the pContainer, similar to STL $iterators$ called $pView$, an abstraction of task graph of computation called $PARAGRAPH$ and an Adaptive Runtime System that includes a communication library, scheduler and perfomance monitor. For detailed information on the STAPL project, please see [8], [27].

In this work, we made use of the STAPL pGraph, one of the STAPL pContainers, as the parallel data structure for representing both the region graph and the roadmap graph. Our proposed method was implemented as a STAPL pAlgorithm.

VI. Experimental Setup and Results

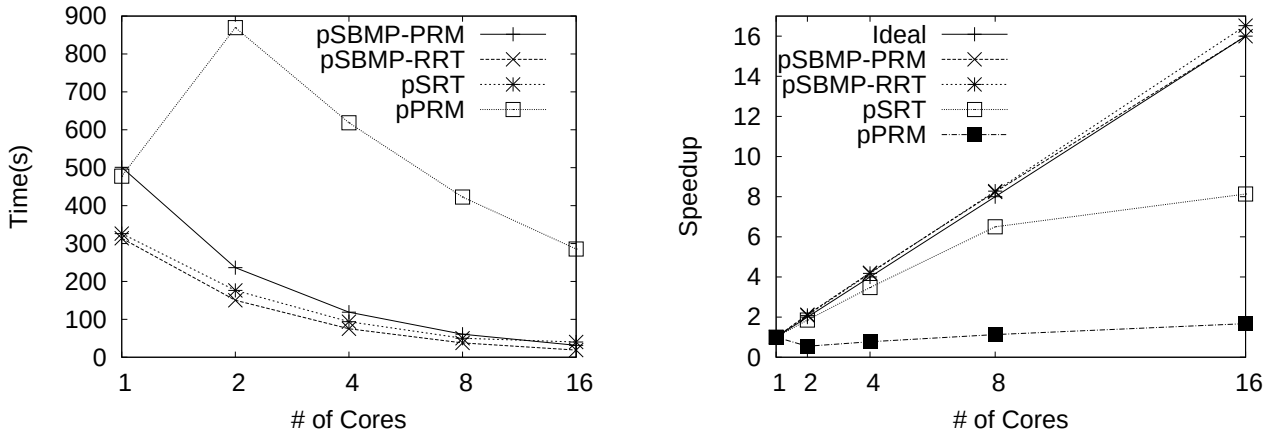
A. Experimental Setup

1) Algorithms: We implemented four different algorithms. The first two were based on our proposed approach but with two different strategies as the underlying sequential planner. These two implementations are referred to as pSBMP-RRT, a parallel sampling-based motion planning method with RRT as the underlying sequential planner, and pSBMP-PRM, a parallel sampling-based motion planning method with PRM as the underlying sequential planner. For evaluation and comparison, we implemented two additional parallel algorithms: the parallel PRM (pPRM) [2] and parallel sampling-based roadmap of trees (pSRT)[1], [22]. Please note that pPRM and pSRT were implemented based on our understanding of how they were described in the literature and it is possible that different implementations may perform better.

2) Environments and Robots: We used two different kinds of environments. The first is a homogeneous cluttered environment with dimensions of 512x512x512 units. The cluttered elements span the x-axis. The cluttered environment has a total of 216 obstacles, each of size 2x64x64 units, as shown in Figure 3(a). The second environment shown in Figure 3(b) is a non-homogeneous cluttered environment. This particular environment models the floor plan of the H.R. Bright building (HRBB), the building that houses the Departments of Computer Science and Engineering and Aerospace Engineering at Texas A&M University.

In both environments, we use two different kinds of robots: a 4x4x4 unit cube-like rigid body robot and a three-link articulated linkage robot, with each link having dimensions of 7x1x1 units.

3) Machine Architectures: Our experiment was carried out on two massively parallel computers. The first is a Cray XE6 petascale machine at Lawrence Berkely National Laboratory. It has 6384 nodes and a total of 153,216 cores with 217 TB of memory and peak performance of 1.288 peta-flops. The second machine is a major computing cluster at Texas A&M University. It has a total of 300 nodes, 172 of which are made of two quad core Intel Xeon and AMD Opteron processors running at 2.5GHz with 16 to 32GB per node. The 300 nodes have 2400 cores in all with over 8TB of memory and a peak performance of 24 Tflops. Our code was written in C++ using the STAPL library [8], [27] and compiled with gcc 4.4.4 on the LINUX cluster and gcc 4.6.1 on the Cray XE6 machine.



(a) Time (b) Speed up

Fig. 4. Comparison of our proposed method (pSBMP-PRM and pSBMP-RRT) to two existing approaches: pPRM and pSRT

B. Experimental Results

1) Comparison with Previous Approaches: We tested the four algorithms (pSBMP-PRM, pSBMP-RRT, pPRM and pSRT) on the LINUX cluster varying the processor count from 1 to 16. The input sample size was fixed at 9600 for each of the four algorithms. Each experiment was run five times and the average maximum time for the 5 runs was computed. Figures 4(a) and (b) show the running time and speedup for the four algorithms. From Figure 4, one will observe that our proposed method (pSBMP-PRM and pSBMP-RRT) achieves good scalability compared to the existing methods. For this particular experiment, we stopped at a processor count of 16 because the two existing algorithms (the pPRM in particular) could no longer scale beyond 16 processor counts.

2) Effects of Different Environments and Machine Architectures: We subjected our method to further experiments in order to observe how it would perform in different environments and machine architectures. Even though these problems exhibit different levels of difficulty and homogeneity leading to differences in running time, we observe that their relative performances are still similar.

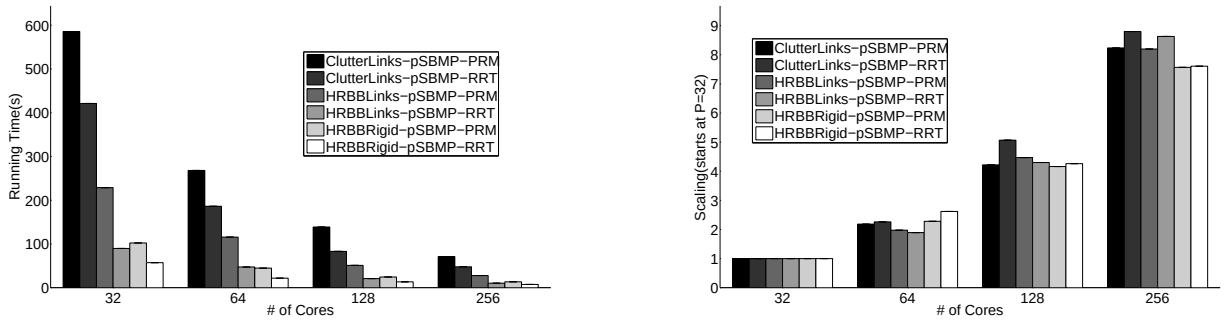
Figure 5 shows both the timing and scalability results for three different motion planning problems. The first problem is the cluttered environment with an articulated linkage robot (ClutterLinks), the second is the building environment with an articulated linkage robot (HRBBLinks), and the third is the building environment with a rigid body robot (HRBBRigid). We observe that the more difficult the problem, the better the scaling. The basic reason for this is that processors (cores) are fully engaged with computation which in some cases (if the algorithm and experiments are properly designed) lowered the overhead cost of idle or inter-processor communication.

We also observe that scalability improves with an increase in sample size. For the same reason as with problem difficulty, increasing sample size ensures that the processors are fully engaged with computation. Figure 6 shows results for varying sample size for the articulated linkage robot in a cluttered environment problem. This set of experiments was carried out on the LINUX cluster with processor counts from 32 to 256.

To study scalability and test the limit of our method, we explore further experiments on a Cray XE6 petascale machine. In this experiment, we tested processor counts of 240, 480, 720, 960 and 1200. The results are shown in Figure 7. We observe that scalability is still possible on a massively parallel machine such as the Cray XE6. The results also suggest that, to the extent possible, our proposed method is independent of machine architecture. Thus, though there may be variance in results, we still expect to see similar performance and scalability across different platforms.

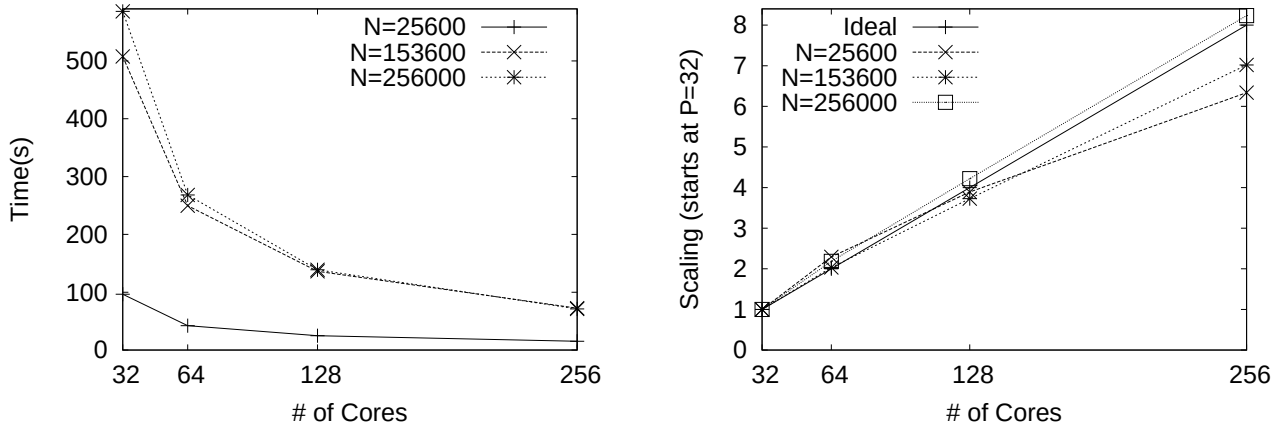
3) Effects of Region Connection on Performance and Roadmap Connectivity: Regional roadmap connection is the last step in our method. At this step, we attempt to connect individual roadmaps to form a full roadmap representing the connectivity of the free space. The connection is limited to adjacent regions. In this initial work, the choice of how to connect was based on the sizes of the connected components (CCs) in each region and the distances between the CCs across neighboring regions. Our implementation also provides a flexible way to connect by allowing the user to specify how many CCs to attempt to connect. We show both the number of CCs and the size of the largest CC before and after region connection as evaluation metrics in Table I.

Our study shows that connection methods and k impact both performance and connectivity of the resulting roadmap. While the region connection running time increases with k, the increase in running time does not significantly degrade performance or affect overall scalability of the method. Table I shows that the time spent at the region connection phase is a fraction of the total time and that much of the time is spent during the roadmap construction phase. Connectivity improvement is measured in terms of decrease in the number of CCs after region roadmap connection as well as increase in the size of connected CCs. A successful



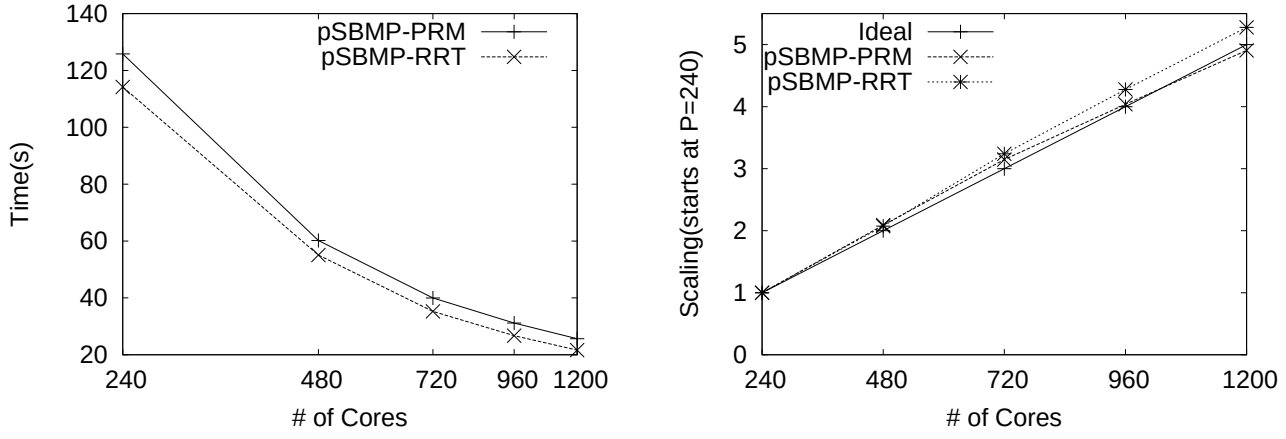
(a) Time (b) Scaling (Processor counts at *P*= 32, 64, 128, 256)

Fig. 5. Results from three different motion planning problems on LINUX cluster using pSMBP-PRM and pSMBP-RRT methods



(a) Time (b) Scaling (Processor counts at *P*= 32, 64, 128, 256)

Fig. 6. Results from varying input size for the articulated linkage robot in a cluttered environment using pSMBP-PRM method



(a) Time (b) Scaling (Processor counts at *P*= 240,480,720,960,1200)

Fig. 7. Higher processor counts on Cray XE6 petascale machine

region connection leads to bigger CCs and possibly a more connected roadmap, especially if the size-based method is used.

VII. Conclusion

In this paper, we describe a scalable approach for parallelizing sampling-based motion planning algorithms. Our framework uses the subdivision of C-space to achieve scalability. We compare a prototype implementation of our method to two existing parallel algorithms for sampling-based motion planning and demonstrate that our approach achieves better and more scalable performance. We presented experimental results using up to 1200 cores on a Cray XE6 petascale machine and up to 256 cores on a LINUX cluster.

TABLE I

REGION CONNECTION PERFORMANCE

k	△ number of CC	△ size of largest CC	Region Connect Time (s)	Total Time (s)
Largest CC Method				
1	31	24414	0.107	37.385
2	61	24513	0.336	37.512
4	122	24570	0.938	37.711
Closest CC Method				
1	21	5982	1.518	38.132
2	54	5672	2.857	39.663
4	124	10425	9.499	47.084

We demonstrated that our framework is flexible enough to support different planning schemes.

Future work will extend our current approach such that we can more efficiently handle more complex environments and attempt higher dimensional problems. Our future work will also include a more detailed analysis of the quality of the roadmap resulting from our method.

Acknowledgement

The authors would like to thank Dezshaun Meeks for his contributions as an undergraduate research intern in our lab during the summer of 2011.

REFERENCES

[1] M. Akinc, K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki. Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In *The International Symposium on Robotics Research (ISRR)*, Sienna, Italy, October 2003.

[2] N. M. Amato and L. K. Dale. Probabilistic roadmap methods are embarrassingly parallel. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 688–694, 1999.

[3] O. B. Bayazit, J.-M. Lien, and N. M. Amato. Better flocking behaviors using rule-based roadmaps. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 95–111, Dec 2002.

[4] O. B. Bayazit, G. Song, and N. M. Amato. Enhancing randomized motion planners: Exploring with haptic hints. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 529–536, 2000.

[5] O. B. Bayazit, G. Song, and N. M. Amato. Ligand binding with OBPRM and haptic user input: Enhancing automatic motion planning with virtual touch. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 954–959, 2001. This work was also presented as a poster at *RECOMB 2001*.

[6] J. Bialkowski, S. Karaman, and E. Frazzoli. Massively parallelizing the rrt and the rrt*. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2011.

[7] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. Int. Conf. Artif. Intel.*, pages 799–806, 1983.

[8] A. Buss, Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, G. Tanase, N. Thomas, X. Xu, M. Bianco, N. M. Amato, and L. Rauchwerger. STAPL: Standard template adaptive parallel library. In *Proc. Annual Haifa Experimental Systems Conference (SYSTOR)*, pages 1–10, New York, NY, USA, 2010. ACM.

[9] S. Carpin and E. Pagello. On parallel rrts for multi-robot systems. In *Proc. Italian Assoc. AI*, pages 834–841, 2002.

[10] H. Chang and T. Y. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1012–1019, 1995.

[11] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[12] T. S. D. Devaurs and J. Cortes. Parallellizing rrt on distributed-memory architectures. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011.

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

[14] D. Hsu, L. Kavraki, J.-C. Latombe, and R. Motwani. Capturing the connectivity of high-dimensional geometric spaces by parallelizable random sampling techniques. In *Proc. IEEE Workshop Randomized Parallel Computing(WRPC)*, 1998.

[15] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2719–2726, 1997.

[16] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[17] Y. Koga, K. Kondo, J. Kuffner, and J. Latombe. Planning motions with intentions. In *Proc. ACM SIGGRAPH*, pages 395–408, 1995.

[18] J. J. Kuffner and S. M. LaValle. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 995–1001, 2000.

[19] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 473–479, 1999.

[20] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, Berlin/Heidelberg, 2005. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Utrecht/Zeist, The Netherlands, 2004.

[21] M. A. Morales A., L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. C-space subdivision and integration in feature-sensitive motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3114–3119, April 2005.

[22] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Trans. Robot. Automat.*, 2005.

[23] E. Plaku and L. E. Kavraki. Distributed sampling-based roadmap of trees for large-scale motion planning. *IEEE Transactions on Robotics and Automation*, 38:793–884, 2005.

[24] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato. (RESAMPL): A region-sensitive adaptive motion planner. In *Algorithmic Foundation of Robotics VII*, pages 285–300. Springer, Berlin/Heidelberg, 2008. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), New York City, 2006.

[25] A. P. Singh, J.-C. Latombe, and D. L. Brutlag. A motion planning approach to flexible ligand binding. In *Int. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, pages 252–261, 1999.

[26] G. Song and N. M. Amato. Using motion planning to study protein folding pathways. In *Proc. Int. Conf. Comput. Molecular Biology (RECOMB)*, pages 287–296, 2001.

[27] G. Tanase, A. Buss, A. Fidel, Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, N. Thomas, X. Xu, N. Mourad, J. Vu, M. Bianco, N. M. Amato, and L. Rauchwerger. The STAPL Parallel Container Framework. In *Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog. (PPoPP)*, pages 235–246, San Antonio, Texas, USA, 2011.

[28] S. Thomas, G. Tanase, L. K. Dale, J. M. Moreira, L. Rauchwerger, and N. M. Amato. Parallel protein folding with STAPL. *Concurrency and Computation: Practice and Experience*, 17(14):1643–1656, 2005.

[29] L. Zhang, Y. Kim, and D. Manocha. A hybrid approach for complete motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 7–14, 2007.