Randomized Motion Planning on Parallel and Distributed Architectures *

Stefano Caselli Monica Reggiani
Dipartimento di Ingegneria dell'Informazione
Università di Parma - 43100 Parma - Italy
{caselli,reggiani}@ce.unipr.it

Abstract

Motion planning is a fundamental problem in a number of application areas, including robotics, automation, and virtual reality. This paper describes a parallel implementation of a motion planning algorithm particularly suited for complex systems characterized by many degrees of freedom. The implementation is based on the concurrent exploration of the search space by a randomized planner replicated on each node of the parallel architecture. All processing elements compete to obtain a solution over the entire search space in an OR-parallel fashion. Reported results refer to a low-cost cluster of PCs and an SGI Onyx2 parallel machine. The experiments emphasize the effectiveness of the approach for complex, high-dimensionality planning problems. We believe that the approach may be useful in other complex search problems, especially when the parallel architecture exhibits relatively high communication latency.

1 Introduction

Motion planning consists in the determination of a collision-free path between given workspace locations in a cluttered environment. The plan may involve one or more movable objects subject to various kinematic and dynamic constraints [18]. This problem plays a fundamental role in a number of application areas, ranging from aircraft maintenance procedures to virtual reality and robotics [9, 12].

Due to the complexity of motion planning, the computation times needed to solve a number of practical applications on sequential computers are incompatible with the specific real-time requirements. One of the approaches investigated to address motion planning of complex systems is based on a random heuristic search of the path guided by a potential function [1, 18]. This approach has brought a remarkable improvement in performance and range of applicability over previous planning techniques and has inspired additional work on motion planning based on random search [16, 20]. A serious drawback of the random search approaches, though, is the high variance in both computation time and quality of the solution found (related to the path length). Furthermore, excessive computation times still preclude motion planning in many applications.

In this paper we describe the parallel implementation of a path-planning algorithm based on the concurrent exploration of the search space by replicated instances of the random path planner. Each processing node receives a complete instance of the planning problem. Performance improvement stems from the increased likelihood of finding a better solution by exploiting multiple independent searches. Owing to its minimal communication requirements, this approach (termed *random competition* in [7]) is also well suited for parallel or distributed architectures characterized by relatively high communication latency, such as Networks of Workstations (NOWs).

The paper is organized as follows. Section 2 briefly summarizes the relevant notions of motion planning. Section 3 describes RPP, the random path planning algorithm exploited in our parallel implementation. Section 4 describes the approach to parallelization based on random competition, whereas Section 5 provides details about the actual implementation. Section 6 presents experimental results obtained on a low-cost cluster of PCs and an 8 processor SGI Onyx2 Infinite Reality. These results refer to an articulated robot with a free base moving inside a 3D environment. Finally, Section 7 provides pointers to related research and Section 8 concludes the paper.

2 The motion planning problem

The general area of motion planning includes several more specific problem cases [14]. In *static* motion planning problems the obstacles are known a-priori, whereas in *dynamic* motion planning knowledge of the environment is built on-line during the exploration process it-

*Partial support for this research has been provided by MURST, Italy (Project MOSAICO, "Design Methodologies and Tools of High Performance Systems for Distributed Applications"), and by ASI, Italy (Project No. ARS-96-182, "Dextrous Manipulation for Space Robot Tasks").

self. In *time-invariant* problems the system whose motion is being planned is the only active entity of the environment, whereas in *time-varying* problems external objects can move as well. In a *multimovers* problem several objects move in a shared environment. Due to our research bias [2], in the following we will refer to the objects whose motion must be planned as the *robots*.

Given the complexity of the general formulation, the following core problem is typically addressed (*basic motion planning problem*):

> *Given an initial configuration and final configuration of the robot in its workspace, find a path, starting at the initial configuration and terminating at the goal configuration, while avoiding collision with the obstacles* [8].

Solving the basic motion planning problem (static knowledge, time-invariant environments) is the key issue also faced in solving the remaining problems, and hence has received the largest attention [14, 18]. The parallel motion planner described in this paper addresses the basic problem as well, even though it can also deal with the multimovers case.

The *workspace* is the physical environment (2D or 3D) where the robot actually moves and where the obstacles are located. The *configuration* of an object with a given shape is the minimum set of independent parameters fully specifying the position of each point of the object. The *configuration space* or *C-space* is the set of all configuration tuples. A *path* is a curve in the configuration space. The number of *degrees of freedom (dof)*, or N_{dof}, is the number of parameters needed to specify the configuration of an object and corresponds to the dimension of the configuration space. N_{dof} strongly determines the complexity of motion planning. Any physical configuration of a robot corresponds to a point in the configuration space. All configurations where the robot would be in contact or compenetrated with an obstacle (or part of itself) correspond to regions to be avoided in the C-space termed *C-obstacles*. *C-free* is the set of all configurations outside C-obstacles.

Motion planning algorithms can be classified into several classes, based upon their generality and the type of guarantees that they provide. *Complete* algorithms guarantee to find a free path between any two given configurations as long as such a path exists, or return the failure otherwise. Due to their complexity, complete algorithms are seldom applicable to practical cases. The main factors affecting motion planning complexity are the dimension of the configuration space (N_{dof}) and the number of obstacles in the workspace. The general problem in C-space of any dimension is PSPACE-hard. PSPACE-hardness results have been later established for several variations of robot path planning problems. A complete algorithm with single exponential time complexity in the number of degrees of freedom is given in [3]. General discussions of the complexity issues are given in [18] and [14].

General complete algorithms have been applied only to problems with $N_{dof} \leq 4$. As most interesting problems are characterized by $N_{dof} \geq 6$, currently the emphasis is toward practical and efficient algorithms, trading off completeness for solution speed. *Probabilistic* algorithms do not compute the geometric representation of the entire free space. Rather, they identify subsets of C-space configurations according to some policy including randomness features and then develop a probabilistic roadmap connecting them [16]. Probabilistic algorithms can be at best *probabilistic-complete*, i.e., they find a free path, if such a path exists, with probability approaching 1 with increasing execution time. An effective *heuristics* exploited in motion planning consists in the definition of a potential field attracting the robot toward the goal. The RPP algorithm at the basis of our work is a probabilistic-complete algorithm exploiting a potential field heuristics to explore the C-space.

The computational cost of a probabilistic motion planning algorithm is difficult to assess. It is often empirically expressed as the wall clock execution time on a set of problem instances. Hwang in [13] attempts a preliminary classification of the difficulty of planning problem instances. Unfortunately, there is no common set of path planning benchmarks allowing fair comparison of the proposed algorithms and their implementations. In this paper we generate motion planning problems of increasing complexity by increasing the number of degrees of freedom of an articulated robot.

3 Random Path Planning

Barraquand and Latombe in [1] proposed a probabilistic-complete algorithm that they successfully applied to problems intractable with complete algorithms. Such algorithm, called Random Path Planning (RPP), is the starting point of this work.

The RPP algorithm constructs a graph whose nodes are the local minima of a potential function U, defined in the discretized configuration space of the robot. Then RPP searches the path connecting the initial and final configurations inside the graph. When the planner finds a path between two local minima, it connects them with an arc of the graph. The potential function U driving the exploration of the configuration space is constructed from a potential that attracts the robot to the goal (placed in the global minimum of U) and a repulsive potential around the obstacles (Figure 1). Starting from the initial configuration q_{init} of the robot, the algorithm executes a *gradient motion*, following the negative gradient of the potential U until it finds a local minimum q_{loc}. If the potential in q_{loc} identifies it as global minimum the algorithm terminates with success returning

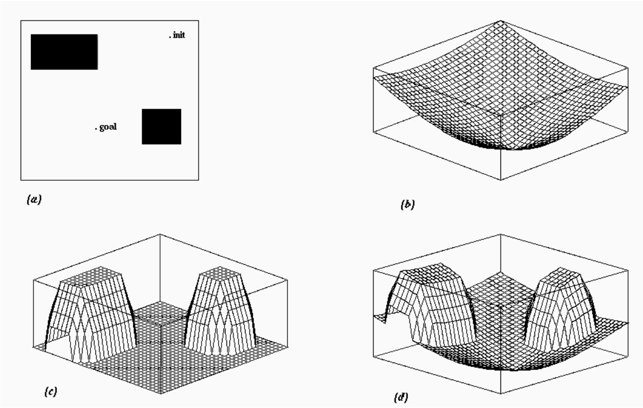

Figure 1. Construction of the potential field in the C-space: starting from a bidimensional C-space containing two C-obstacles (a), the potential field (d) is obtained from the combination of an attractive field to the goal (b) and of a repulsive field (c).

the path; otherwise the algorithm tries to exit from the local minimum in which has been trapped. In order to escape from the domain of attraction of the local minimum the algorithm executes several *random motions* from q_{loc}. Every random motion is immediately followed by a *gradient motion* that will find a new minimum. The algorithm therefore proceeds to the incremental construction of the graph until it finds the goal configuration.

In order to reduce the computational cost of the algorithm, in certain phases random choices are executed. During the gradient motion all the neighbors of the configuration would have to be explored so that the configuration with the lowest potential can be chosen as successor. Unfortunately, the number of neighbors (equal to $3^{N_{dof}} - 1$) is too large, and therefore the neighborhood is completely explorable only with few degrees of freedom. For examples with $N_{dof} = 10$ every configuration has approximately 60000 neighbors. In order to improve the efficiency, in this phase only a randomly chosen subset of the neighbors is explored. Owing to this approximation, it is not assured that the algorithm follows the direction of the negative gradient of U during the exploration of the configuration space. Another source of randomness is represented by the behavior of the algorithm when it tries to escape from the attraction domains of the local minima with random motions.

Even though the formulation of the algorithm is independent from the potential function U, U largely influences RPP performance. In particular, if the function U does not introduce many local minima, the gradient motion has more chances to reach the global minimum; furthermore, if every local minimum has a small attraction domain, it is more likely to successfully escape from the local minimum with a random motion [11].

4 Parallel motion planning

Given the complexity of motion planning, the goal of a parallel approach is to decrease the time neeeded for the solution of a given problem or to improve the quality of the solution. The fundamental phase of the RPP algorithm is the construction and search in the graph of the local minima. Efficiency reasons have already mandated the use of “quasi best first” search techniques, with sequences of random motions and gradient motions constructing and exploring only the part of the C-space needed to obtain a solution. The parallel algorithm maintains these characteristics while also trying to subdivide among the processors the load of the path search in the work space.

The various approaches to parallel graph search in the literature can be classified into two fundamental models. In the first one the search space is subdivided among the processors, that communicate to exchange the knowledge acquired during the exploration [17]. In the second model, based on OR parallelism, processors are independent and search for a solution over the entire graph [15, 7]. Both models incur into an overhead due to parallel programming. In the first model the major source of overhead is the time spent in order to transfer the data among the processors. The efficiency of the algorithm is therefore affected by the effectiveness of the available communication protocols. In the OR-parallel model the major source of overhead is represented by the computations of the independent processors exploring the same zone of the graph. The choice between the two models should be carried out trying to decrease the overhead, and therefore taking into account the architecture on hand and the cost of the computational resources.

The parallel approach described in this work has been conceived for a low cost NOW platform built with networked PCs. This kind of system provides computational power at low cost thanks to the software support for communication among the processing nodes. In spite of the recent developments in network technology and communication protocols, standard libraries such as MPI or PVM show a high communication latency. This remark has motivated the choice of a search approach based on OR parallelism, where each node executes a probabilistic planning on the whole workspace. This technique, called *random competition* [7], exploits a set of independent parallel processors executing the same program with different initial search seeds. The task to be solved is not partitioned among the processors, but rather every processor receives an identical copy of the entire problem and the execution is stopped when one of the processors finds the solution.

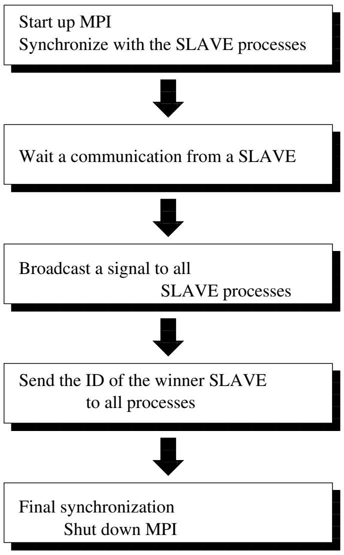

Figure 2. Operation of the Master process.

5 Parallel implementation

A master process coordinates the execution of the overall planning task by interacting with a number of slave processes. The operation of the master process is shown in Figure 2, whereas the behavior of the slaves, each running on a separate processing node, is shown in Figure 3. Thus, parallel path planning consists of the following phases:

1. **startup phase**: The main process (master) transmits the entire task to all the slave processes, and sets up the randomness of the algorithm in order to differentiate the search space of the slaves.

2. **working phase**: Every slave process starts the solution of the entire problem with the RPP algorithm, and executes random actions in the course of the exploration of the configuration space. A slave process interrupts its search in case of success or when it receives a signal from the master.

3. **termination**: If a slave process identifies a solution of the problem or it fails in the execution (because it does not find a solution within the predefined time), it communicates to the master the result of its own execution. After an arbitration among the received messages, the master interrupts all the slave processes and returns the final result.

This approach always improves the solution time of the problem when random algorithms are used or when different algorithms are available with a hierarchy in time that depends upon the characteristics of the problem. In these situations the execution times are variable and it is impossible

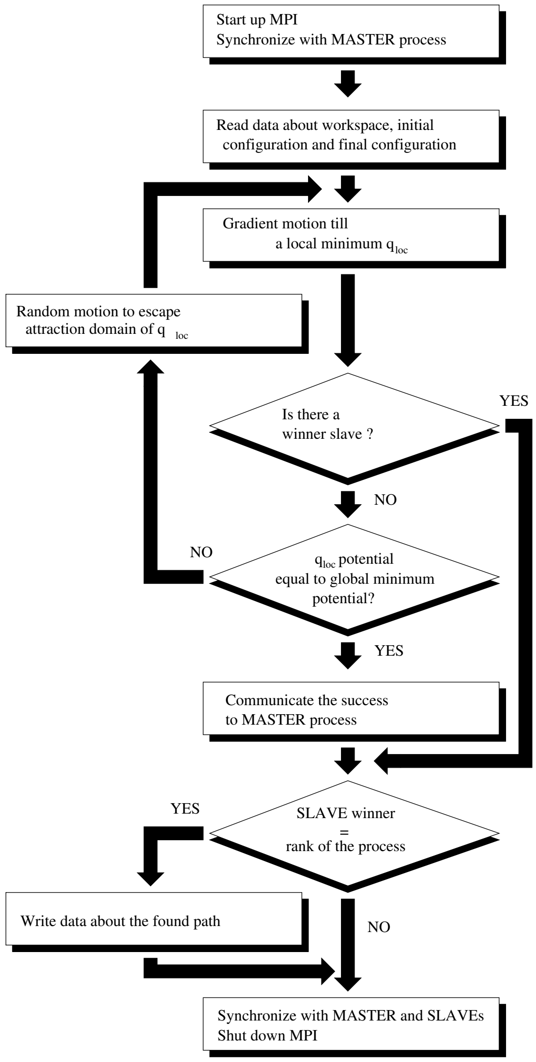

Figure 3. Operation of a Slave process.

to establish in advance which algorithm or which initialization seed will yield the best performance. Moreover, since every process is able to solve the entire problem, even if one of the processes fails the remaining ones can still find the solution. The random competition is also flexible and simple to implement, since the addition of one processor does not imply changes to the code. As in the random competition the communication among the processors is limited to the startup and termination phases, the approach introduces a minimal communication overhead and hence is well suited for NOW architectures. We term our parallel path planning implementation as *PRPP, Parallel Randomized Path Planning* algorithm.

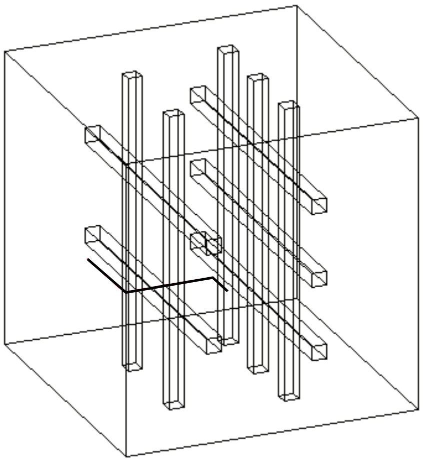

Figure 4. 3D-representation of the workspace used in the experiments. The articulated robot is shown with darker segments.

Parma²								
dof	**1**	**2**	**3**	**4**	**8**	**12**	**16**	**20**
7	20.5	14.75	11.9	12.2	11.5	10.9	10.7	10.6
9	265.9	129.7	99.4	73.6	56.8	42.6	34.6	35.5
11	532.6	302.5	247.5	194.3	104.4	73.4	70.0	54.1
Onyx2								
dof	**1**	**2**	**3**	**4**	**8**	**12**	**16**	**20**
7	4.4	3.1	2.9	2.8	2.8	2.5	2.4	2.4
9	50.6	34.1	20.2	21.8	13.6	9.3	8.1	7.2
11	141.5	87.9	57.9	54.1	26.5	20.7	18.3	12.6

Table 1. Average solution times (in seconds) on 100 executions on the two systems.

6 Experimental results

In this section the results obtained with a low-cost NOW platform and a commercial parallel machine are discussed. Parma^2 (the University of *Parma Par*allel *Ma*chine) [6] is a NOW platform of six PCs (two PPro200 and four Pentium 100) interconnected to a Fast-Ethernet hub with 12 ports[1]. Message passing is supported by the LAM/MPI environment [19]. For the experiments described in this work only the four Pentium 100 were used, in order to be able to evaluate the speedup on a set of homogeneous machines. The Silicon Graphics Onyx2 InfiniteReality is a parallel machine comprising eight 4MB cache MIPS R10000 processors and 1 GB of memory per processor pair. This system is based on an Origin CC-NUMA architecture with 650MB/sec memory bandwidth and 711 MB/sec interprocessor communication bandwidth. Software support for messages passing is provided by the MPI package implemented by SGI [21]. This implementation is based on the MPICH package but is optimized for Origin2000 and Onyx2 machines.

To account for the high variance of randomized path planning, all PRPP data reported in this section are the average time of 100 executions.

In the random competition approach all processors search the solution independently and the communication time is a negligible fraction of the total time. Owing to the minimal communication and synchronization requirements, the time required to find a solution with a set of n identical slaves is given by the shortest among n execution times with only one slave. Therefore it is possible to evaluate the temporal behavior of the PRPP algorithm on more processors than those physically available. This approach has been validated by executing identical experiments on clusters of real and virtual processors with the same number of nodes. This test has indeed resulted in identical performance results.

Following the approach described above, when the experiments required more processors than those physically available the execution time was determined as the minimum time among a set of n sample executions on a single slave.

The path planning experiments described in this paper refer to the 3D workspace shown in Figure 4. This workspace includes an articulated robot with a free base (a snake-like robot) which is required to navigate to a goal location while coping with the grid-like obstacles. The environment is discretized in an array of 128^3 cells. When the number of degrees of freedom of the robot moving inside the workspace increases, the difficulty of the problem increases as well, so it is straightforward to construct a series of path planning problems of increasing complexity.

The average solution times obtained with the Parma^2 and Onyx2 platforms (Table 1) decrease when the number of slave processors increases. The decrease rate is not constant. In fact, after a phase of decrease the average solution time reaches an asymptotic value (Figure 5).

Of course, the speedup exhibits a similar trend: after a phase of increase it also reaches an asymptotic value, as shown in Figure 6. It is possible to predict the trend and the asymptotic value of the speedup S as:

$$S(k) = \frac{E(T_1)}{E(T_k)}$$

where $E(T_k)$ is the expected value of the execution time with k slave when a set of sample executions with only one slave is available. Supported by elementary probability theory, Ertel has shown that:

$$\lim_{k \to \infty} S(k) = \frac{E(T_1)}{t_0}$$

[1] This was Parma^2 configuration at the time of the tests reported in this paper. Parma^2 has since been significantly upgraded and expanded [6].

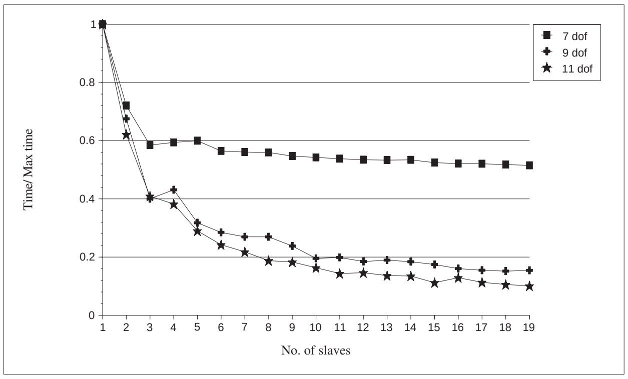

Figure 5. Ratio of average time to maximum average time of three problems with different complexity on the Onyx2 system.

Parma^2								
dof	**1**	**2**	**3**	**4**	**8**	**12**	**16**	**20**
7	1.09	1.51	1.86	1.83	1.95	2.04	2.09	2.1
9	0.99	2.02	2.64	3.56	4.61	6.15	7.57	7.39
11	1.24	2.19	2.68	3.41	6.35	9.04	9.47	12.25
Onyx2								
dof	**1**	**2**	**3**	**4**	**8**	**12**	**16**	**20**
7	0.9	1.3	1.35	1.39	0.89	1.58	1.62	1.63
9	1.2	1.78	3.01	2.79	4.47	6.53	7.52	8.43
11	1.16	1.86	2.83	3.03	6.18	7.92	8.98	13.02

Table 2. Speedup on the two systems.

where t_0 is the minimum execution time of random search [7].

The number of slave processes where the asymptotic value of average execution time and speedup are reached depends upon the complexity of the problem. In fact when the number of the degrees of freedom of the robot increases, the dimension of the configuration space also increases. More slaves are then required in order to explore the graph of the local minima in a time close to the minimum execution time t_0. When this value is reached, any further increase in the number of slave processes does not increase the probability to explore a part of the configuration space not yet involved in the search. At this point the slaves only perform redundant work and the efficiency decreases (Figure 7).

As shown in Figure 8, the speedup does not depend upon the architecture where the program is executed. The strong imbalance in communication performance (two orders of magnitude difference in interprocessor communication bandwidth between Onyx2 and Parma^2), in fact, does not affect the scaling behavior of PRPP. Indeed, with the random competition approach communications is limited

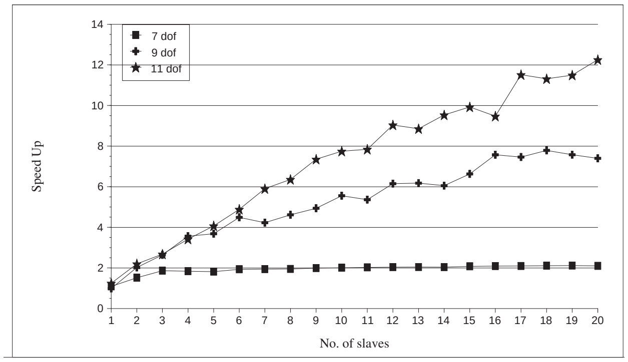

Figure 6. Speedup with three problems of different complexity on the Parma^2 system.

Parma^2								
dof	**1**	**2**	**3**	**4**	**8**	**12**	**16**	**20**
7	1.09	0.76	0.62	0.46	0.24	0.17	0.13	0.1
9	0.99	1.01	0.88	0.89	0.58	0.51	0.47	0.37
11	1.24	1.1	0.89	0.85	0.79	0.75	0.59	0.61
Onyx^2								
dof	**1**	**2**	**3**	**4**	**8**	**12**	**16**	**20**
7	0.9	0.65	0.45	0.35	0.11	0.13	0.1	0.08
9	1.2	0.89	1	0.7	0.56	0.54	0.47	0.42
11	1.16	0.93	0.94	0.76	0.77	0.66	0.56	0.65

Table 3. Efficiency on the two systems.

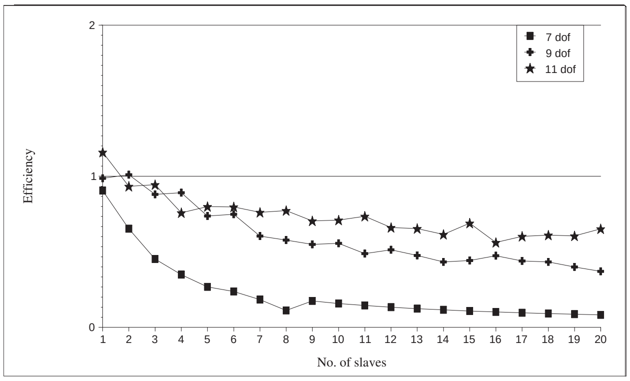

Figure 7. Efficiency of three problems with different complexity on Onyx2 system.

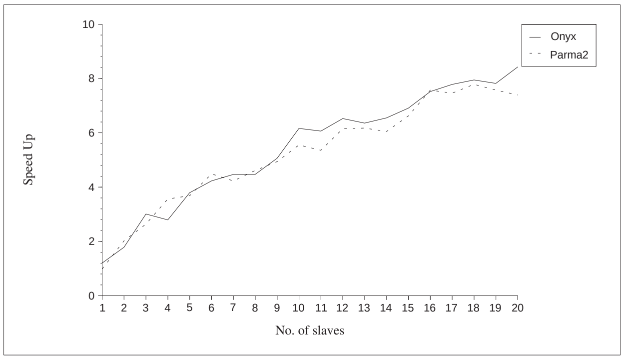

Figure 8. Comparison between the speedup of the Onyx2 and Parma^2 systems on the 9 dof problem.

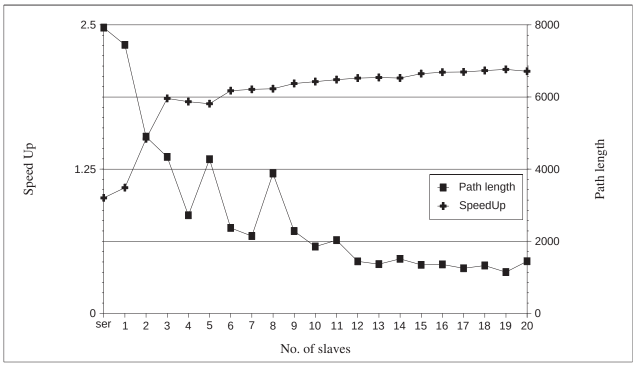

Figure 10. Path length and Speedup for a robot with 7 degrees of freedom.

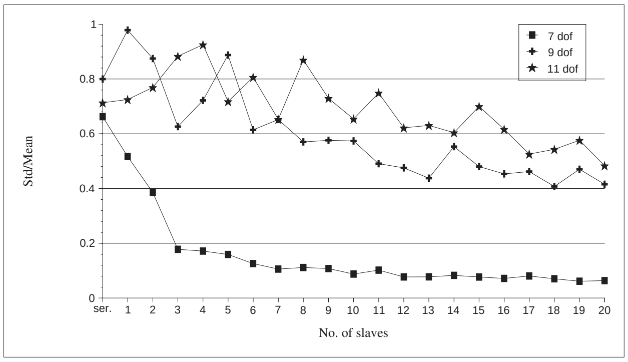

Figure 9. Ratio of Standard Deviation to Average Time for three problems with different complexity on the Parma^2 system.

to the transmission of the initialization seed and the notification of the winner slave to each slave. The better absolute performance figures of the Onyx2 are simply due to the fastest processors and the higher memory bandwidth.

Initial tests with the sequential algorithm on a 333MHz PentiumII show that the execution times take about one half in input/output phases and one fourth in computationally intensive phases of the equivalent times of a 100MHz Pentium. Therefore the evolution of low cost architectures already allows distributed architectures with performance close to those of parallel machines when the parallel approach has low communication overhead.

Although the increment in the number of slaves does not yield lower *average* solution times when the maximum speedup value is reached, this increment may still be useful because the *standard deviation* of the execution time is decreased (Figure 9). Moreover, the experimental results demonstrate that also the quality of the solutions (i.e., the path length) improves when the number of processors increases. Figure 10 shows that even the quality of the solution found reaches a limit (as should be expected), but this limit is reached with more processors than those required to obtain the maximum speedup. The possibility to obtain a better solution with less variance in computation time can suggest the use of more processors than those needed to obtain the minimum average execution time.

7 Related work

Recently, several authors have investigated parallel techniques to address motion planning problems. An extensive list of parallel approaches to motion planning is given in [10]. Among them, closest to the proposal of this paper are the work of Gini and others [5, 4], that introduces a planner based on a parallel formulation of the algorithm proposed in [1], and the one in [20], whose planner generates various random subgoals and then tries to connect them in parallel with the initial and final configurations.

From a general viewpoint, the parallel technique described in this work has been proposed in [15] and applied by Ertel for theorem proving problems in [7].

8 Conclusions and future work

This paper has described PRPP, a parallel motion planning algorithm suited for complex systems. The implementation is based on the concurrent exploration of the search space by a randomized planner replicated on each node of the parallel architecture. All processing elements compete to obtain a solution over the entire search space in an OR-parallel fashion. Reported results refer to a low-cost cluster of PCs and a commercial CC-NUMA parallel machine. The experiments emphasize the effectiveness of the approach for complex, high-dimensionality planning problems. The motion planner described in this paper has also been exploited in a robot kitting task involving a 6 dof manipulator fed with parts by a mobile robot [2].

We believe that the approach may be useful in other complex search problems, especially when the parallel architecture exhibits relatively high communication latency. We now plan to investigate variations of the approach calling for tighter cooperation among the processors during the search phase.

References

[1] J. Barraquand and J.-C. Latombe. Robot motion planning: a distributed representation approach. *Internation Journal of Robotics Research*, 10(6), Dec. 1991.

[2] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli. A real–time library for the design of hybrid robot control architectures. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS '98*, Victoria, BC, Canada, Oct. 1998.

[3] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, 1988.

[4] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May 1995.

[5] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993.

[6] Computer Engineering Group University of Parma. Parma^2 Home Page. `http://www.ce.UniPR.IT/research/parma2`.

[7] W. Ertel. Performance analysis of competitive OR–parallel theorem proving. Technical Report FKI16291, Technische Universitat Munchen, Germany, Nov. 1991.

[8] K. Gupta. Pratical motion planning: An overview and state of the art. In *Workshop on Pratical Motion Planning in Robotics: Current Approaches and Future Directions*, Minneapolis, MN, Apr. 1996.

[9] K. Gupta and A. P. del Pobil. *Workshop on Pratical Motion Planning in Robotics: Current Approaches and Future Directions*. IEEE International Conference on Robotics and Automation, Minneapolis, MN, Apr. 1996.

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

[11] D. Hsu, L. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Workshop on the Algorithmic Foundations of Robotics*, Houston, TX, Mar. 1998.

[12] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE International Conference on Robotics and Automation*, Albuquerque, NM, Apr. 1997.

[13] Y. K. Hwang. Completeness vs. efficiency in real applications of motion planning. In *Workshop on Pratical Motion Planning in Robotics: Current Approaches and Future Directions*, Minneapolis,MN, Apr. 1996.

[14] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys*, 24(3), Sept. 1992.

[15] V. K. Janakiram, D. P. Agrawal, and R. Mehrotra. A randomized parallel backtracking algorithm. *IEEE Transactions on Computers*, 37(12), Dec. 1988.

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

[17] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. Benjamin/Cummings Publishing Company, 1994.

[18] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Press, 1991.

[19] Ohio Supercomputer Center. *MPI Primer/Developing with LAM*. The Ohio State University, Nov. 1996.

[20] C. Qin and D. Henrich. Randomized parallel motion planning for robot manipulation. Technical Report Computer Science 5/96, University of Karlsruhe, Germany, Feb. 1996.

[21] Silicon Graphics Inc. *Onyx2 Technical Report*, Oct. 1996.