Parallel Search Algorithms for Robot Motion Planning*

Maria Gini
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

Abstract

We show how paths for dexterous robots can be generated in a few seconds or less using parallel informed randomized search on multicomputers. The experimental results we present have been obtained for a variety of robots with six or more joints operating in realistic 3D workspaces.

1 Introduction

Motion planning is the process of computing paths that will allow a robot to move to different positions in its environment without hitting obstacles. Many algorithms have been developed [Latombe, 1991], but most are never used in practice because of their computational complexity [Hwang and Ahuja, 1992].

The intent of this paper is to show that plans for multi-jointed dexterous robot arms which operate in realistic environments can be synthesized very quickly by parallel algorithms. The ability to plan paths quickly is important to make motion planning useful in application areas, such as industrial robotics, teleoperation [Lumelsky and Cheng, 1993], control of redundant robots [Laliberte and Gosselin, 1994]. Real-time motion planning coupled with real-time sensing will allow robots to adapt their planned paths to take into account the uncertainties of the real world. Even more important, it will allow robots to react to unanticipated events and to quickly replan their paths whenever needed.

2 Motion Planning

Research in the area of robot motion planning can be traced back to the late sixties, but most of the work has been carried out more recently. Over the last few years the theoretical and practical understanding of the issues has increased rapidly, and a variety of solutions have been proposed. Latombe [Latombe, 1991] provides an extensive overview.

There are two different spaces associated with motion planning algorithms, the workspace and the configuration space (C-space). The workspace is the world that the robot must move through, the C-space is the set of all robot configurations. The dimensionality of the C-Space is the number of parameters required to fully specify a configuration of the robot. The search is done in C-space because the robot becomes a point in C-space. The motion planning problem then becomes one of computing a decomposition of the C-space and searching through sequences of cells to find a path that involves no collisions with obstacles.

*This work was supported in part by Contract numbers DA/DAAH04-93-G-0080 and DA/DAAH04-93-G-0131 between the Army Research Office and the University of Minnesota for the Army High Performance Computing Research Center. Additional support was furnished by NSF/CDA-9022509, NSF/CCR-9405380, and the Center for Advanced Manufacturing, Design and Control of the University of Minnesota.

To obtain acceptable performance, some methods do a significant amount of preprocessing of the configuration space (C-space) [Kavraki, 1994], or place landmarks in C-space that are then used by a local planner [Bessiere *et al.*, 1993, Chen and Hwang, 1992]. Other methods make assumptions on the type of robot (for instance, [Adolphs and Tolle, 1992] takes advantage of the symmetry of the workspace with respect to the first axis of the robot), or use a coarse discretization of C-Space. Real time has been achieved in detection of imminent collisions [Shaffer, 1992, Wikman *et al.*, 1993], but not for path planning.

In an effort to decrease the computation time, some researchers have devised parallel methods. Lozano-Perez [Lozano-Perez, 1991] was the first to develop a parallel algorithm to compute the discretized C-space for the first three links of a six *dof* manipulator. This method works well, but it is limited to relatively coarse C-space discretizations due to memory limitations. A genetic based approach has been implemented using 128 T800 transputers with excellent performance [Bessiere *et al.*, 1993]. The method places landmarks in free space until it is able to generate a path using local methods.

Our method is a parallel formulation of the Randomized Path Planner proposed by Barraquand and Latombe [Barraquand and Latombe, 1991]. Space is represented with bitmap arrays. The configuration space is discretized and searched using heuristic search with random walks to escape local minima. The C-space is searched but not stored, because of memory reasons. For example, assume a robot arm with k joints, where each degree of freedom is quantized into n discrete levels. Such an arm has a C-space consisting of n^k unique configurations. If we assume $n = 90$, an arm with four joints ($k = 4$) has over sixty five million configurations, and an arm with six joints ($k = 6$) has over five hundred billion configurations. Even if it were possible to compute all of the C-space, the amount of memory required to store it would be prohibitive.

There are many reasons for our selection of the algorithm to parallelize. First, parallel formulations of randomized search can easily be developed using randomized allocation schemes. Second, the grid-based representation of the workspace is especially convenient when sensors are used to construct it, as shown, for instance, by [Moravec, 1988], [Laliberte and Gosselin, 1994].

We have shown that our parallel formulation is capable of generating plans in very short time frames on various parallel architectures [Challou, 1995], including a 1024 processor nCUBE2[1], a 512-processor CM5[2], and a 16-processor network of Sun workstations.

One might argue that massively parallel machines are not a viable platform for path planning systems due to their prohibitive cost. However, due to the continuing progress in VLSI design and economy of scale resulting from their widespread use, the cost of the processors which massively parallel machines utilize is expected to decrease. When this occurs, it will be feasible to build large scale parallel computers at a relatively small cost. Hence, it is reasonable to believe that massively parallel machines will be readily available within the next decade. Shared memory architectures with a few processors, and networks of workstations are already widely available, even in industrial settings. Recent advances in networks will soon make parallel computing on clusters of workstations a viable option for high performance computing.

3 Parallel Motion Planning

Many methods have been developed for searching a state-space graph or tree. We categorize them into the two broad classes proposed by Langley [Langley, 1992] called *systematic* and *nonsystematic search.*

Systematic search methods enumerate each node in the search space according to a particular strategy. Systematic search methods produce each possible path only once, thus they minimize the amount of redundant work.

Nonsystematic search methods, often called randomized search methods, select a node at random at each choice point in the search and record it as a step in the path. This process is repeated until a solution is found or a depth limit is reached. Nonsystematic search methods do not retain a complete memory of states

[1] nCUBE2 is a registered trademark of the nCUBE corporation

[2] CM-5 is a registered trademark of the Thinking Machines Corporation. The results obtained on the CM-5 that are presented in this paper are based upon a beta version of the software and, consequently, are not necessarily representative of the performance of the full version of the software.

they have previously generated and expanded, and thus may generate the same paths more than once when searching for a solution. Nonsystematic search methods are only probabilistically complete (and thus cannot report that a problem has no solution with perfect certainty), but are capable of outperforming uninformed systematic schemes, such as depth-first search, on certain problems. Nonsystematic searches are particularly effective on tasks whose solutions are many and deep, and though purely randomized search methods can be successful, they can often benefit from heuristic knowledge [Langley, 1992].

To date, most parallel implementations have focused on systematic search methods. Two types of parallel formulations have been developed: (1) communication-based formulations that partition the search space among the processors, and (2) formulations in which each processor explores the entire search space randomly with no interprocessor communication. The latter formulations are commonly referred to as randomized search methods. In both formulations the first processor to find a solution sends a termination signal to the remaining processors, and then reports its solution. Randomized schemes have been shown to outperform uninformed methods under certain criteria [Mehrotra and Gehringer, 1985]. Randomized search methods can often benefit from heuristic knowledge [Langley, 1992].

To implement the randomized motion planning algorithm on parallel architectures, we broadcast the workspace bitmap and desired goal location to all processors, and check for a message indicating that a processor has found a solution. Each processor runs the same basic program. The only interprocessor communication is the initial broadcast and the termination check. Randomized search and random walks are the means by which the search-space is partitioned, as they probabilistically insure that each processor searches different parts of C-space.

An outline of the algorithm is shown down here. The * at the end of some of the steps indicates a point where the algorithm checks for a termination message from other processors. If a termination message has been received, the algorithm terminates its computation.

Step I: Compute the heuristics used to guide the search.
Step II: Search using the heuristics :

```
repeat until goal found or global time-out
    Gradient Descent until local minimum *
    while no improvement or time-out
        repeat K times or until improvement found
            Random Walk to escape local minimum *
            Gradient Descent until a local minimum *
        if no improvement
            then Randomly Backtrack
        if improvement found
            then append new path to previous path
if goal found
    then broadcast termination message
```

Consider the search starting at the start robot configuration. The gradient descent forces the search in the direction of the goal node that appears closest. If the heuristic is misleading then, at some point, every successor is worse than the current node. When this occurs, random search with a randomly chosen depth bound is executed. We call this step a random walk. Gradient descent resumes from the state at which the random walk terminates. The sequences composed of a random walk followed by gradient descent search are repeated for a fixed number (K) of trials or until a better node is found. If, after K trials, no better node has been found, then backtracking is performed to a randomly picked point in the solution path. The cycle of random walks followed by gradient descent is then resumed. When a better node is found the new part of the path found is appended to the previous path and the process resumes with a new gradient descent step.

The idea behind the random walks and randomized backtracking is to find a place in a different region of the search space where the heuristic is more reliable. In that event the gradient descent search can quickly descend toward a goal configuration. Successors of a node are generated in a random manner until a successor

is found that has a better heuristic value than the current configuration. Thus, the first legal successor with a better value than its parent is adopted as the next step in the path.

A brief theoretical explanation for the success of parallel randomized allocation schemes is as follows. Let $P_1(t)$ be the probability that a single processor will find a solution within time t, and let $P_k(t)$ be the probability that a k-processor parallel randomized allocation scheme will find a solution within time t. Let the random variable T_1^i be the time it would take processor i to find a solution, if allowed to run to completion. Since this is equivalent to running multiple trials on a single processor, the T_1^i's are independent and identically distributed. The probability $1 - P_k(t)$ that the solution time on k processors will exceed t is just the probability that none of the k processors will find a solution within time t:

$$1 - P_k(t) = (1 - P_1(t))^k \tag{1}$$

To interpret this formula, suppose a single processor has only a 10% probability that it can solve the problem within a given time $t_{10\%}$. Then a 32 processor system has over a 96% probability of finding a solution within $t_{10\%}$, and a 64 processor system has over a 99% chance of doing so. Space does not permit a more detailed analysis, which can be found in [Challou, 1995].

We define speedup as:

$$S = \frac{E[T_1]}{E[T_k]}. \tag{2}$$

where $E[T_1]$ is the average uniprocessor solution time, and $E[T_k]$ the average k processor solution time. If $E[T_k]$ is less than $E[T_1]$, then, on average, the k processor randomized allocation formulation will deliver speedup over the uniprocessor algorithm.

If $E[T_k]$ is less than $\frac{1}{k} \cdot E[T_1]$ then, on k processors, the parallel randomized allocation formulation will yield, on average, superlinear speedup over the uniprocessor algorithm. This is because on k processors the first processor to find a solution stops all the others, so there is no need to wait for solutions that take a long time. On a single trial on a uniprocessor a bad choice made early might significantly delay the completion of the search. On k processors a bad choice made by one processor does not prevent the other processors from making a better choice.

Our parallel formulation of randomized heuristic search has proved extremely effective in solving motion planning problems. The formulation has yielded impressive performance gains on every problem we have employed it on, sometimes delivering superlinear speedup [Challou, 1995].

We have observed superlinear speedup when using a small number of processors in the majority of our experiments. This shows that it pays to use our parallel algorithm, even when only 4 or 8 processors are available. The speedup decreases and eventually becomes sublinear as the number of processors is increased.

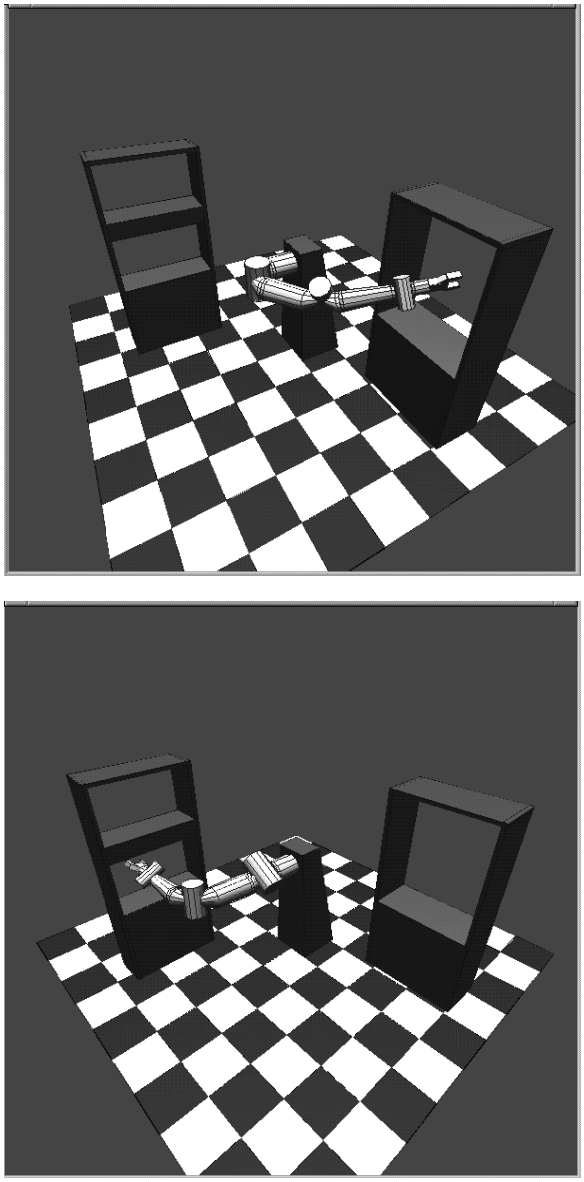
4 Experimental results

We have performed a variety of experiments with multiple robot arms, different configurations of obstacles, and using different parallel architectures.

4.1 Experiments with a Robotics Research K-1207i arm

For the problem shown in Figure 1 just 16 processors are required to cut the average solution an order of magnitude to under ten seconds, and 64 processors cut the average solution time to under five seconds [Challou *et al.*, 1995b].

In addition to delivering paths in shorter time frames, another important property of the parallel formulation is that, when it is executed with a larger number of processors, it tends to produce better solutions. In the example, 32 processors yield a solution path length about one fourth as long as the average solution path length delivered by one processor, and 128 processors reduce the average solution path length by an order of magnitude. The variance in time to solution behaves similarly, that is, it falls off as the number of processors attempting to solve the problem increases.

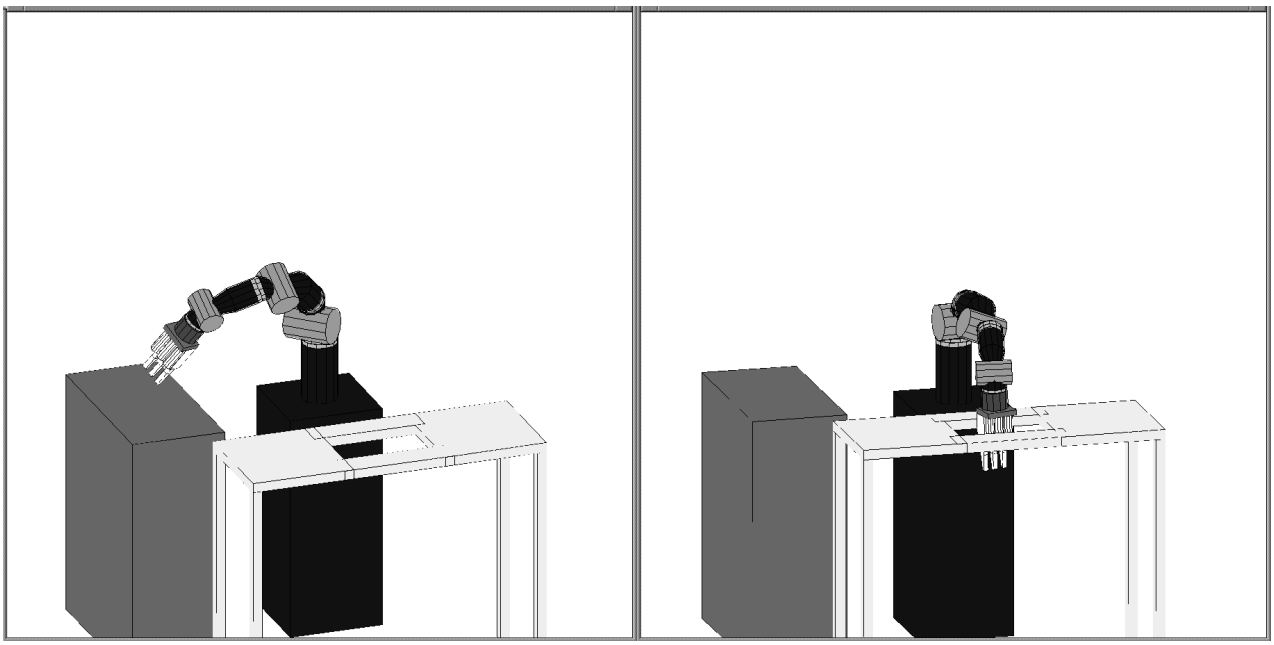


No Processors	1	4	8	16	32	64	128	256	512
Avg Search Time	127.77	26.75	16.63	9.62	6.01	4.81	2.78	2.08	1.59
Std Dev	142.00	20.92	10.41	7.02	3.71	2.69	1.53	0.78	0.33
Avg Path Length	8618	6211	5361	4550	2558	2776	1660	1179	722
Std Dev	5425	4487	4566	4189	2356	1989	1207	746	295
Speedup	1.00	4.78	7.68	13.28	21.26	26.56	45.96	61.43	80.36
Efficiency	1.00	1.19	0.96	0.83	0.66	0.42	0.36	0.24	0.16

Figure 1: Start and Goal Configurations for a Robotics Research K-1207i 7–*dof* arm operating in a 128^3 cell workspace. The robot is reaching from the cabinet on the right into the cabinet on the left. The table shows data for at least 64 runs on a CM-5 multicomputer. All times are in seconds, path lengths in number of steps.

4.2 Experiments with a 7-degree of freedom arm

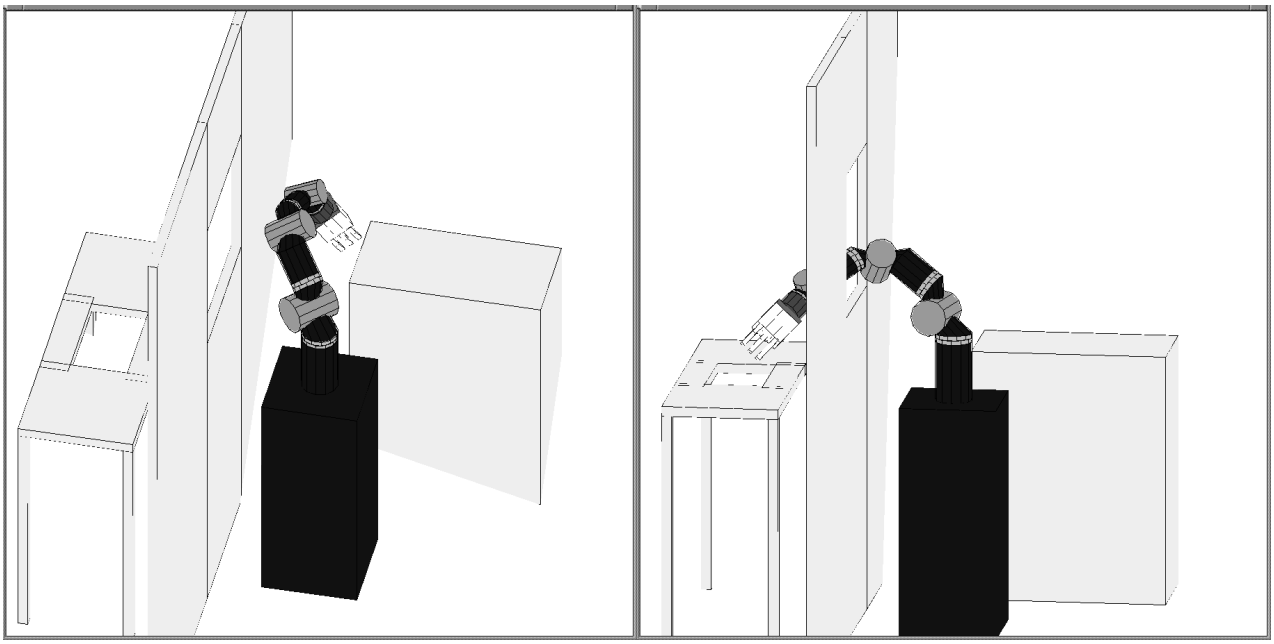
For the problem illustrated in Figure 2, just 32 processors are required to insure path synthesis in an average time of under six tenths of a second, and, in our trials, only 128 processors are necessary to insure synthesis of a solution path in under one second. As the number of processors looking for a solution path increases, the average solution path length decreases as well. Only 32 processors are required to cut the average solution path length by over a factor of ten, and 128 processors deliver an average solution path length more than two orders of magnitude less than the average sequential solution path.



No Processors	1	32	128	256	512
Avg Search Time	18.63	0.56	0.39	0.29	0.27
Std Dev	23.33	0.25	0.10	0.07	0.05
Avg Path Length	1589	120	72	59	66
Std Dev	3653	86	14	21	25

Figure 2: Start and Goal Configurations for a 7 Degree of Freedom arm operating in a 128 x 128 x 128 cell workspace. The robot is reaching from a table on the left down and through an opening in the table in front of it. The table summarizes the data for at least ten runs of the random planner on up to 512 processors of a CM-5 multicomputer. All times are in seconds.

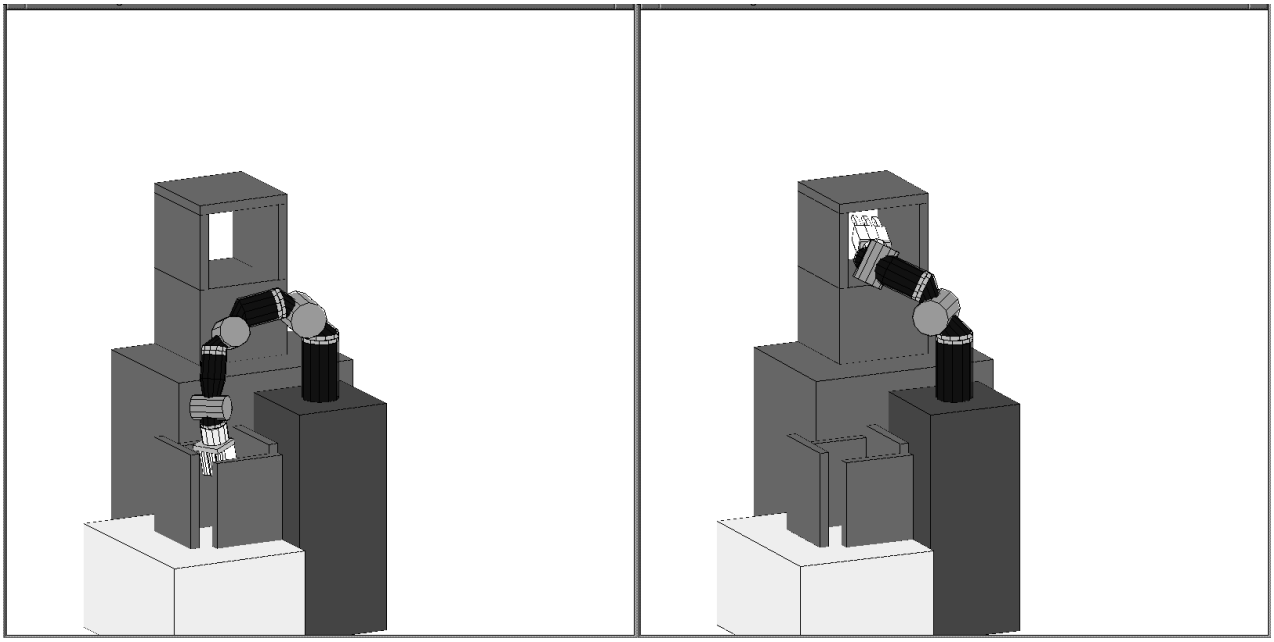
For the problem illustrated in Figure 3, just 32 processors are required to cut the average solution time to under ten seconds, and 128 processors deliver solutions ten times faster than the sequential version of the system (on average) [Challou *et al.*, 1995a]. In our tests, 512 processors insured that the solution was found in under three and one half seconds. Moreover, 32 processors cut the average solution path length by an order of magnitude, and 128 processors cut the worst case solution path length by one order of magnitude compared to the average sequential case.



No Processors	1	32	128	256	512
Avg Search Time	60.94	8.20	4.05	3.73	2.47
Std Dev	36.36	3.90	1.63	1.34	0.74
Avg Path Length	11427	7390	3409	3488	1937
Std Dev	5228	3360	1600	1551	950

Figure 3: Start and Goal Configurations for Seven Degree of Freedom Robotics Research Arm operating in a 128 x 128 x 128 cell workspace. The robot is reaching from the table behind it, through an opening in the wall on its left, and down to the table with the hole in it. The table summarizes the data for at least ten runs on up to 512 processors of a CM-5 multicomputer. All times are in seconds.

For the problem illustrated in Figure 4, again just 32 processors are required to cut the average solution time to under ten seconds. In our tests, 256 processors insured that the solution was found in under two and one half seconds.



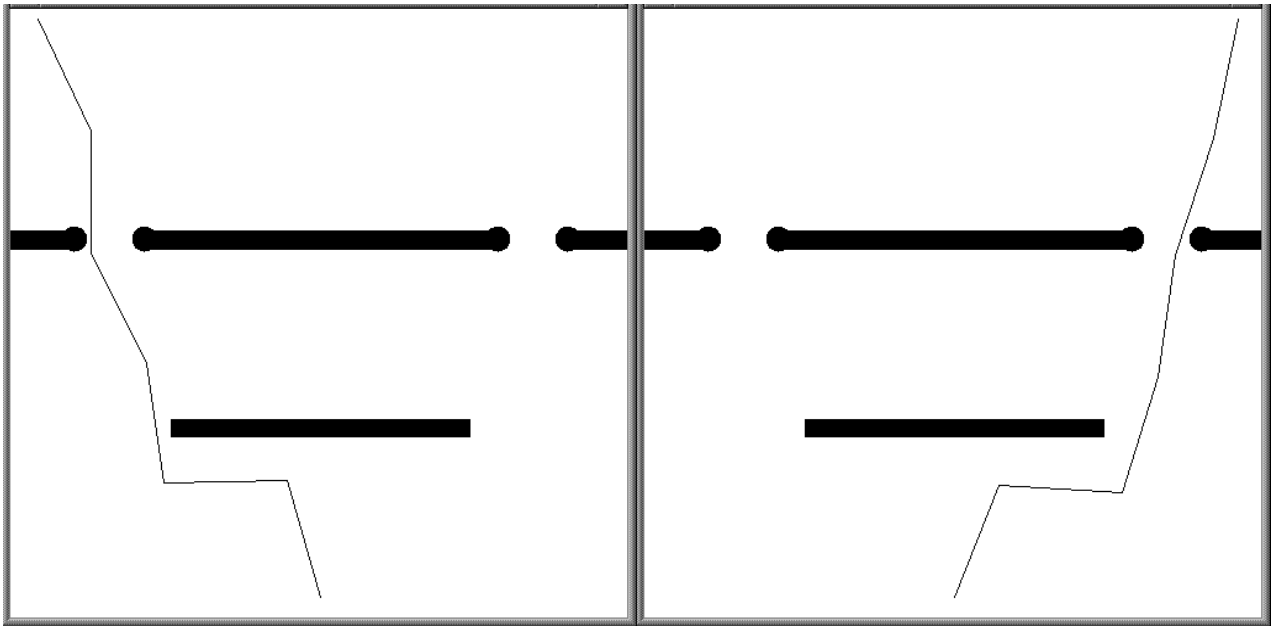
No Processors	1	32	64	128	256
Avg Search Time	102.34	8.39	5.36	3.37	2.32
Std Dev	108.33	5.24	3.26	2.17	1.26
Avg Path Length	4264	1178	1351	967	531
Std Dev	5196	1550	1942	1277	476
Speedup	1.00	12.02	19.09	30.37	44.11

Figure 4: Start and Goal Configurations for a 7 Degree of Freedom arm operating in a 128^3 cell workspace. The robot is reaching from the box in front of it, up and into the box on the left. The table show data for at least 64 runs on a CM-5 multicomputer. All times are in seconds.

4.3 A very difficult example

Figure 5 shows the start and goal configurations for one of our test cases involving a six degree of freedom planar robot with one control point operating in a 256 x 256 cell workspace [Challou *et al.*, 1993]. Each joint has up to 256 discrete positions.

The algorithm reduces the average computation time from 34411 seconds (over 9 hours) on one processor to an average of 180 seconds (3 minutes) on 1024 processors. Furthermore, from the average times calculated in Table 5, it is apparent that the algorithm does not require a large number of processors to make significant reductions in the time required to solve this problem instance - just 64 processors are required to solve the problem in an average time of about 22 minutes.



No Processors	1	2	4	8	16	32	64	128	256	512	1024
Avg Search Time	34411	16924	34886	12378	3397	6526	1370	1125	384	289	180
Max Search Time	178824*	164898*	76426	23676	24601*	12967	3050	1919	528	584	223
Min Search Time	28565	16924	10683	4083	716	2636	194	145	198	53	110
Speedup	-	-	4	11.27	41.07	21.38	101.85	124.03	363.39	482.85	775.24

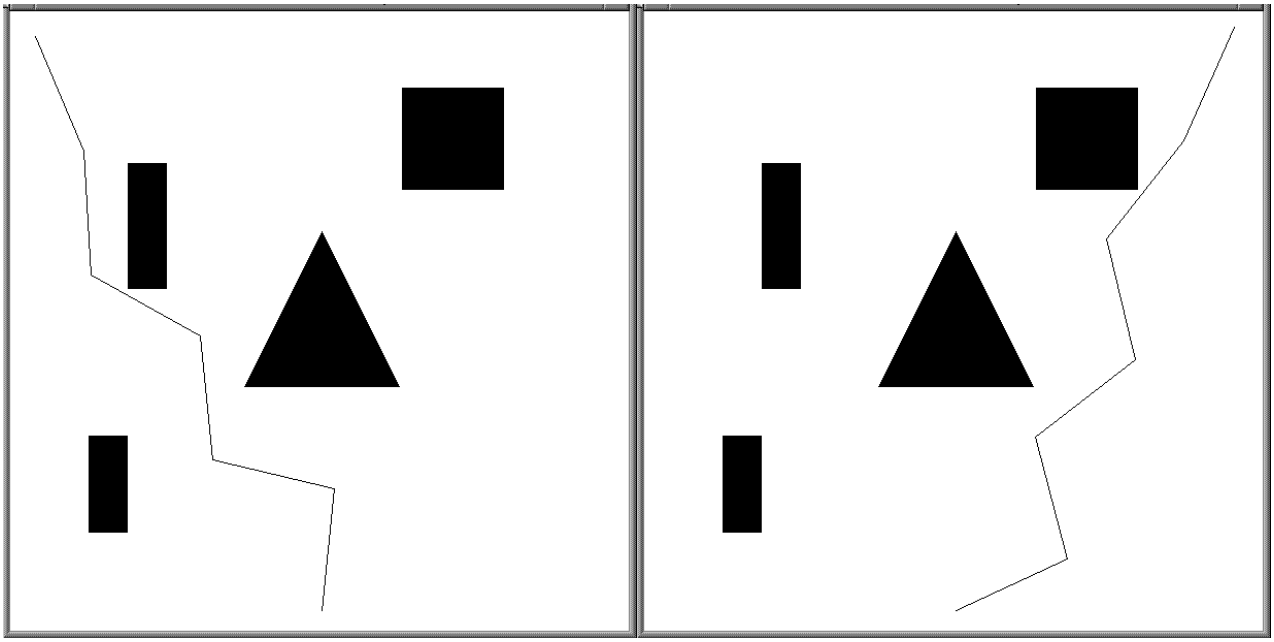
Figure 5: Start and Goal Configurations for Six Degree of Freedom Robot operating in a 256 x 256 cell workspace. Each C-space axis has 256 possible discrete positions. The table summarizes the data for five runs of on up to 1024 processors on nCUBE2 multicomputer. A * indicates no solution was found on that run. All times are in seconds.

The speedup is not calculated for the results up to and including four processors for the following reason. For runs of the planner on up to four processors, the planner failed to find a solution on as many or more runs than it found one on. If we had let the planner run long enough, it would have arrived at a solution since it is probabilistically complete. However it might have taken a great deal more time than allowed by the cutoff bound of about two days that we set Such cases would make the speedup appear a great deal

better than does assuming linear speedup on four processors. For our speedup calculation then, we used four times the average time taken by four processors as the time on which speedup is based.

Given this conservative assumption then, it is interesting to note that speedup oscillates between slightly sublinear and superlinear until it starts to fall off at about 512 processors. Moreover, the time variance required to solve the problem decreases as the number of processors used to solve the problem increases. For example, on sixteen processors, the maximum and minimum time to solve the problem varies by 23885 seconds (or over 6.6 hours), while on 256 processors and up the maximum time difference is 531 seconds (a little less than 9 minutes). Moreover, the planner was unable to formulate a solution to the problem in its worst case behavior on sixteen processors in the time it was allocated (24601 seconds or about 6.5 hours).

4.4 Comparison on different platforms



No Processors	1	2	4	8	16	32	64	128	256	512
nCUBE2	8959	5002	1247	1103	397	204	59	39	32	27
CM-5	3543	1208	315	334	232	39	32	29	19	14
NWS	3088	793	567	247	180	NA	NA	NA	NA	NA

Figure 6: Start and Goal Configurations for Six Degree of Freedom Robot operating in a 256 x 256 cell workspace. Each C-space axis (joint) has 256 possible discrete positions. The base of the robot is fixed in the bottom-center in each frame of the picture. The table summarizes the data for runs on different multicomputers. All times are in seconds.

The table 6 summarizes the data for ten runs on up to 512 processors on an nCUBE2, 512 processors of a CM-5, and a Network of 16 Sun Workstations. The table shows the average time to find a solution on each of the different hardware platforms for the problem instance involving the six degree of freedom fixed-base robot arm operating in a 256 x 256 cell workspace pictured in fig. 6. Each C-space axis has 256 possible discrete positions. Entries labelled NA indicate that timings are not yet available for that number of processors. All times are in seconds.

At first one might be surprised that such a straight forward parallel algorithm fares as well as it does, reducing the average computation time on the CM-5 from almost 1 hour on one processor to an average of 14 seconds on 512 processors, and from almost 1 hour to 3 minutes on a network of 16 workstations.

Acknowledgements

We would like to acknowledge Daniel Boley, Ben Camel, Daniel Challou, Vipin Kumar, and Curtis Olson for their contributions to the research described here; Jean Claude Latombe at Stanford University for providing access to implementations of the Random Path Planner; David Strip and Robert Benner at Sandia National Laboratories for providing access to the nCUBE2.

References

[Adolphs and Tolle, 1992] P. Adolphs and H. Tolle. Collision-free real-time path-planning in time varying environment. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 445–452, 1992.

[Barraquand and Latombe, 1991] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *Int'l Journal of Robotics Research*, 10(6):628–649, 1991.

[Bessiere *et al.*, 1993] Pierre Bessiere, Juan-Manuel Ahuactzin, El-Ghazali Talbi, and Emmanuel Mazer. The Ariadne's Clew algorithm: Global planning with local methods. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 1993.

[Challou *et al.*, 1993] D. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, volume 2, pages 46–51, 1993.

[Challou *et al.*, 1995a] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for for robot motion planning problems. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, volume 1, pages 709–714, 1995.

[Challou *et al.*, 1995b] D. Challou, M. Gini, V. Kumar, and C. Olson. Very fast motion planning for dexterous robots. In *Proc. IEEE International Symposium on Assembly and Task Planning (ISATP)*, pages 201–206, 1995.

[Challou, 1995] D. Challou. Parallel search algorithms for robot motion planning. Ph.D. dissertation, The University of Minnesota, 1995.

[Chen and Hwang, 1992] Pang C. Chen and Yong K. Hwang. SANDROS: a motion planner with performance proportional to task difficulty. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2346–2353, 1992.

[Hwang and Ahuja, 1992] Y.K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[Kavraki, 1994] L. Kavraki. Randomized preprocessing of C-space for fast path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2138–2145, 1994.

[Laliberte and Gosselin, 1994] Thierry Laliberte and Clement Gosselin. Efficient algorithms for the trajectory planning of redundant manipulators with obstacle avoidance. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2044–2049, 1994.

[Langley, 1992] Pat Langley. Systematic and nonsystematic search strategies. In *Proc. Int'l Conf. on AI Planning Systems*, pages 145–152, College Park, Md, 1992.

[Latombe, 1991] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publ., Norwell, MA, 1991.

[Lozano-Perez, 1991] T. Lozano-Perez. Parallel robot motion planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1000–1007, 1991.

[Lumelsky and Cheng, 1993] V. Lumelsky and Edward Cheng. Real-time collision avoidance in teleoperated whole sensitive robot arm manipulators. *IEEE Trans. Systems, Man, and Cybernetics*, SMC-23(1):194–203, Jan/Feb 1993.

[Mehrotra and Gehringer, 1985] R. Mehrotra and E. F. Gehringer. Superlinear speedup through randomized algorithms. In *Proc. Int'l Conf. on Parallel Processing*, pages 291–300, 1985.

[Moravec, 1988] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[Shaffer, 1992] Clifford A. Shaffer. A real-time robot arm collision avoidance system. *IEEE Trans. Robotics and Automation*, RA–8(2):149–160, 1992.

[Wikman *et al.*, 1993] Thomas S. Wikman, Michael Branicky, and Wyatt S. Newman. Reflexive collision avoidance: a generalized approach. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, volume 3, pages 31–36, 1993.