Towards Parallel Real-time Trajectory Planning

Štěpán Kopřiva and David Šišlák and Michal Pěchouček

Abstract This paper exploits the computing power of widely available multi-core machines to accelerate the trajectory planning by parallelisation of the search algorithm. In particular we investigate the approach that schedules the workload on the cores using the hashing function based on the geographical partitioning of the search space. We use this approach to parallelize the AA* algorithm. In our solution, each partition of the geographical space is represented as an agent. The concept is evaluated on the simulation of real-time trajectory planning of aircraft respecting the environment and real aircraft performance models. We show that the approach decreases the planning time significantly on common multi-core machines preserving the quality of the trajectory provided by AA* algorithm.

1 Introduction

In trajectory planning for air traffic control, as well as in other real domains, runtime of the planning process is a crucial parameter. Currently, it is widely believed that the future processors will have more computational cores per microchip instead of higher clock rates. The promising approach to the speed-up of planning process is therefore parallelisation of the computation. In the trajectory planning problem for air traffic control the planned trajectory is required to be close to an optimal one, therefore the A* planner has been widely adapted and many improvements have been proposed. The AA* algorithm [8] uses an adaptive planning step and advanced similarity checking of the states. To further speed-up the planning process, we have decided to parallelise the AA* algorithm.

Štěpán Kopřiva and David Šišlák and Michal Pěchouček
Czech Technical University, Faculty of Electrical Engineering, Department of Computer Science and Engineering, Agent Technology Center,
Email: kopriva@agents.fel.cvut.cz, sislak@agents.fel.cvut.cz, pechoucek@agents.fel.cvut.cz

Several approaches to the parallelisation of the A* algorithm have been proposed. The most straightforward approach is to implement open and closed lists in the shared memory which is used by all the threads performing the search. To maintain data consistency, access to these lists must be synchronised using mutual exclusion locks (mutexes) or concurrent implementations based on the non-blocking algorithms. As a consequence, each time a thread needs to acquire a new node from the open list or check whether a node has been processed before, it has to wait for the other threads to finish their updates on the lists. Since these lists are updated after each expansion, such an approach suffers from a significant synchronisation overhead leading to a performance which is often inferior to the serial A* [1].

To reduce such an overhead, other approaches use a distributed representation of open and closed lists, where each thread handles a part of them. Evett et al. [3] propose an algorithm called Parallel Retracting A* (PRA*), which assigns newly generated states to threads according to a simple representation-based state hashing scheme.

Burns et al. [1] extend the algorithm with a state abstraction mechanism. The new algorithm called abstraction-based PRA* (APRA*) uses a user supplied abstraction function to partition the search state space graph into so-called nblocks that tend to encapsulate highly connected parts of the graph. The nblocks are assigned to the search threads that perform expansion of its states. Since most of the newly expanded states belong to the same nblock, the abstraction reduces the amount of communication and synchronisation needed to perform the search. On 4 threads, they report 1.8x speed-up over the serial A* algorithm in the grid path finding domain.

Another extension of PRA*, called HDA*, was introduced by Kishimoto et al. [4]. In their proposal, synchronous messaging between the threads in PRA* is replaced by asynchronous communication. Their work reports that the algorithm performs significantly better than the original PRA* algorithm. The experimental data in the domain of grid path finding from Burns et al. [2] show that HDA* achieves 1.3x speed-up on 4 threads over APRA*.

An algorithm that combines both abstraction and asynchronous communication, named AHDA*, has been studied by Burns et. al. [2]. The results of experiments performed in classical planning domains (grid path finding, sliding piles and STRIPS planning) suggest that AHDA* outperforms both APRA* and HDA*. On 4 threads, in the domain of grid path finding, AHDA* yields 2.5x speed-up over the serial A* algorithm.

The planning problem addressed in this paper is nearly optimal trajectory planning of an aircraft in the realistic environment of the National Airspace System (NAS) of the United States. The flight trajectory planner has to provide only trajectories which can be further executed (flown) by an airplane with a complex model. The model of the airplane is described by a set of differential equations with many constraints, e.g. there is limited acceleration, cruise speed, pitch angle. For the details about this planning problem see [7].

2 Accelerated A*

The AA* algorithm is introduced to show computation complexity of the flight trajectory planning and thus its direct influence to the distributed simulation. The path planning causes the simulation very nonuniform in two ways. First, the path planning is run in a single moment consuming a lot of resources and is idle for the most of the time. Second, a lot of airplanes are planning trajectory in some sector while not in others. More detailed information can be found in [7, 6].

The AA* algorithm extends the original A* algorithm to be usable in large-scale environments without forgetting about the search precision. The AA* removes the trade-off between the speed and the precision by introducing of the *adaptive sampling*. During the expansion, child states are generated by applying vehicle elementary motion actions using elements' adaptive parametrization. A set of elementary motion actions is defined by the model of the non-holonomic vehicle movement dynamics. The adaptive parametrization varies and thus the algorithm makes larger steps when the current state is far from obstacles and restricted areas and smaller steps when it is closer. The Figure 1 shows the advantage of the adaptive sampling of the AA*. The adaptive sampling can be seen as different size of the step (distance of the arc in the Figure) depending on the distance to the obstacle.

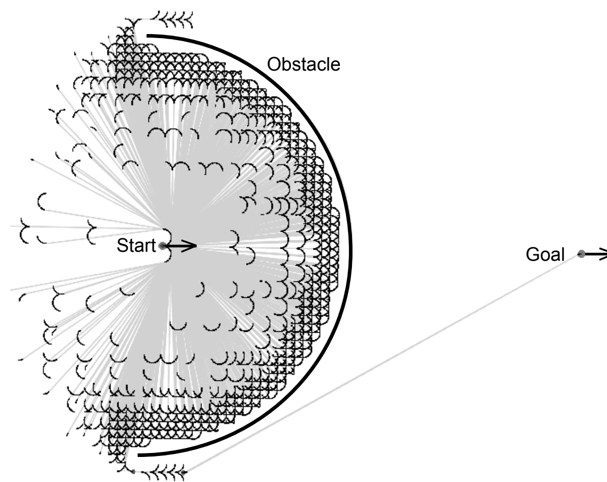

Fig. 1 The adaptive sampling example in the two-dimensional setup. The adaptive sampling can be seen as different size of the step (distance of the arc) depending on the distance to the obstacle.

There is a defined *search precision* specifying the minimal sampling grid step which is used in the areas closest to obstacles. The search precision is defined so that the AA* algorithm does not skip any existing gap between obstacles larger than this precision. Specifically, the AA* algorithm uses the highest possible parametrization

which ensures that the distance to the closest obstacle is not smaller than the distance corresponding to two respective sampling steps.

The adaptive sampling in the AA* algorithm requires a different definition of identity tests when working with OPEN and CLOSE lists. The original equality implementation is replaced by a similarity check. Two states are similar if their Euclidean distance and their direction vector variation is less than a threshold derived from the respective sampling parametrization. Otherwise, the adaptive sampling of a non-holonomic vehicle trajectory causes an infinite state generation in the continuous space. To remove effects of varying sampling, each path candidate generated during the search is smoothed.

Properties of the AA* concept were evaluated on a set of two and three-dimensional setups. The original A* algorithm with a distance-to-target heuristics was chosen as a comparator because it is the only one which provides an optimal solution and does not require any pre-processing of the environment definition. The AA* algorithm provides acceleration of the path planning up to 1400 times in comparison with the original A* algorithm. Moreover, it was found that the AA* algorithm also accelerates the result in case of failure (the path does not exist) due to the reduced number of all generated states.

3 Parallel AA*

The Parallel Adaptive A* (PAA*) algorithm combines the ideas from the HDA* algorithm [4] (distribution of open and closed lists and asynchronous communication) with fast AA*(adaptive sampling) and partitioning of the geographical space suitable for motion planning.

The PAA* algorithm uses distributed open and closed lists. Each core/processor P has a partition of the search space assigned to it and instances of open and closed lists based on the hash function described later. When the planning process starts, the processor which has the starting state in its assigned partition starts the search space exploration. The PAA* algorithm comes with three extensions to AA*.

1. **Incoming Buffer Check.** After the node classification and processing, the algorithm checks whether there are any states in the incoming buffer. For each node from the incoming buffer its presence in the closed list is checked. If the result of the test is negative, the state is put into the open list.
2. **Solution Propagation.** If the processor P_i finds the goal state, it propagates the value of the cost of the trajectory of the final node of the solution to other processors. These processors remove from their open lists states with and continues with the search until the open list is empty. When all processors finish the search, the solution with the lowest cost is applied.
3. **Node Classification.** For each newly generated state on the processor P_i the algorithm checks whether it belongs to P_i using the hash function defining the parallelisation partitioning. If the node doesn't belong to P_i, it is asynchronously sent to the processor that it belongs to.

3.1 Planning Space Partitioning

Assignment of the newly generated nodes to the processors is based on the geographical partitioning of the space. The partitioning of the space affects the number of messages sent among processors, processor utilization and therefore the performance of the algorithm. The design of the optimal partitioning of the geographical space is not in the scope of this paper. The paper just explore whether PAA* algorithm can achieve speed-up effect even though it will use simple partitioning like is presented in Figure 1. The optimal partitioning depends on the number and performance of processors, start and goal positions and on the position and shape of the obstacles. The criterions for this optimization task are the number of messages exchanged among the cores and utilization of the cores. We want to reach the minimal number of messages and the maximal utilization of each core doing the parallel state space exploration. We also intend to balance the load of the processors based on performance.

The partitioning method used for the initial study divides the geographical space based on the start and goal positions to i uniform rectangles, based on the number of processors. One example of the partitioning of the space for i=3 is depicted in Figure 2. The mapping of the processors to the partitions is assigned dynamically in the beginning of the planning task.

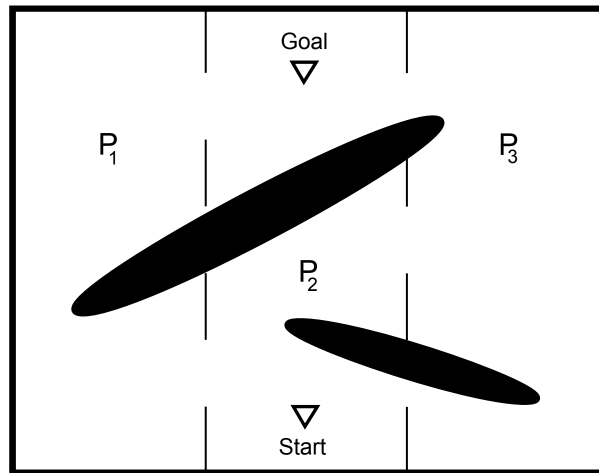

Fig. 2 Dividing the space using the hash function

4 Experiments

We have experimentally evaluated the PAA* planning algorithm using five artificial scenarios and several real scenario. We have implemented PAA* in Java using the multi-agent testbed AgentFly [5]. For this short paper we omit the experiments on the computer cluster and grids as it is expected that the algorithm is deployed on board of the planning unit (aircraft). The testing multi-core machine was a dual quad-core 2.66 GHz Xeon E5430 with 6MB L2 cache (total 8 cores) and 8 GB RAM. The original AA* algorithm has been naturally chosen as the reference algorithm. We have measured the run-time of the algorithm and the speed-up.

	AA*	PAA* 2 cores		PAA* 4 cores		PAA* 6 cores		PAA* 8 cores	
	run-time	**run-time**	**speed-up**	**run-time**	**speed-up**	**run-time**	**speed-up**	**run-time**	**speed-up**
A1	238	201	1.184	130.76	1.82	120.20	1.98	103.03	2.31
A2	590	495	1.192	335.22	1.76	302.56	1.95	292.07	2.02
A3	1390	1188	1.169	803.46	1.73	695	2.00	640.55	2.17
A4	1430	1222	1.172	803.37	1.78	729.59	1.96	668.22	2.14
A5	2314	2012	1.151	1345.34	1.72	1151.24	2.01	1096.68	2.11
R1	762	680	1.12	577	1.32	564.44	1.35	540.3	1.41
R2	980	867.25	1.13	765.62	1.28	742.42	1.32	715	1.37
R3	840	717.94	1.17	646.15	1.30	636.36	1.33	604.3	1.39

Table 1 Artificial scenario and real-world scenario results. Run-times are measured in milliseconds. Each measurement is the average from 10 runs.

4.1 Artificial Scenario Setup

In the artificial scenarios, we have inserted from one to five obstacles in such a way that each obstacle spans 3/4 of the width of the geographical space. For the configuration with one obstacle, this obstacle is placed exactly in the middle of the distance between the start and goal positions of the planning. The width of the obstacle is 1/20 of the distance between the start and goal positions and the length of each obstacle is 3/4 of the width of the geographical space. For the configuration with two obstacles, the parameters of both obstacles are the same as in the case of one obstacle, but the placement of the obstacles is different. One obstacle is placed in 1/3 of the distance from the start position to the goal position and the second one is placed in 2/3 of the distance. Moreover, the first obstacle is shifted to the very right side of the geographical space and the second one is shifted to the very left side. The number of obstacles in different configurations of the scenario varies from one to five. The configuation for three obstacles is depicted in Figure 3.

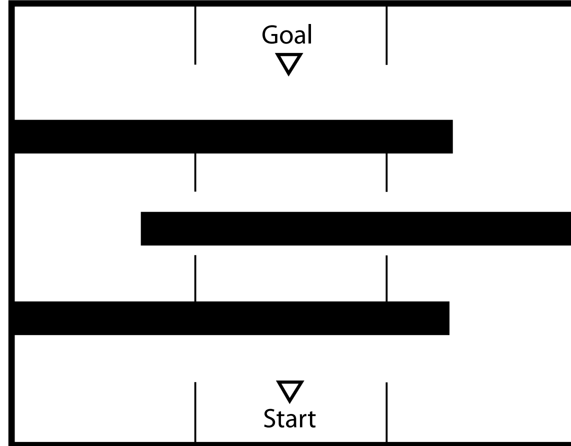

Fig. 3 Artificial Scenario Setup for three obstacles (A3).

4.2 Real-world Scenario Setup

For the real world scenario, we use simulated flights among the cities in U.S. NAS. In this scenario we simulate the en-route phase of flights considering the real airplane flight characteristics based on the Base of Aircraft Data (EUROCONTROL 2009) model. We also consider real obstacles - Special Use Airspaces (SUA), where no aircraft may be present at any time and also minimal distance from ground surface. This scenario simulates the real domain. The selected flights (and thus real scenario configurations) are the following ones: R1: Orlando - Cleveland, R2: Washington - Seattle, R3: Sacramento - Atlanta.

4.3 Results

The results of both artificial and real-world scenarios are presented in the Table 1. The artificial scenarios are denoted as A1 - A5 in the table, where the number in the scenario name corresponds with the number of obstacles in the scenario. The real scenarios are denoted as R1 - R3 and the scenario configuration is described above. For each scenario we have measured run-time, speed-up and the length of the final trajectory. The table presents only the run-time and speed-up values because both AA* and PAA* algorithms provides exactly the same length of the resulting trajectories in the same configuration. The speed-up is computed as

$$speed-up = \frac{run-time(AA^*)}{run-time(PAA^*)}.$$

5 Discussion

The overall average speed-up for artificial scenarios is 1.173 for 2 cores, 1.762 for 4 cores, 1.98 for 6 cores and 2.15 for 8 cores. The speed-up is very similar for each scenario version and is remarkable compared to the real scenario.

For the real scenarios, the average speed-up is 1.14 for 2 cores, 1.30 for 4 cores, 1.33 for 6 cores and 1.39 for 8 cores.

Comparing artificial and real scenarios, it is apparent that the speed-up of the algorithm is dependent on the specific domain configuration - the size of the obstacles, their placement and primarily on the partitioning of the search space. For the used partitioning, the algorithm has got significantly better results on the artificial scenarios. The difference is caused by the obstacle setting - obstacles in the artificial scenarios force the search to spread on all cores, which is not the case in the real scenario. In the real scenarios the main computational load is on the cores that are assigned to the partitions in the middle of the geographical space and the other cores are not utilized.

The Figure 4 presents the dependency of the speed-up on the number of cores. It is obvious that the speedup of the PAA* algorithm doesn't scale up linearly. For the artificial scenarios, the algorithm scales-up well from 2 cores to 4 cores. The performance improvement from 4 cores to 6 cores and then to 8 cores is not so significant, yet important. For the real scenarios the performance improvement from 2 cores to 4 cores is even slower. The performance improvement from 4 cores to 8 cores is low.

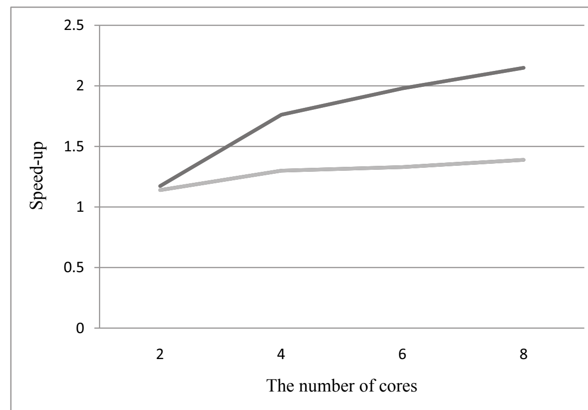

Fig. 4 speed-up for both scenarios, artificial scenario is in black and real-world scenario is in grey.

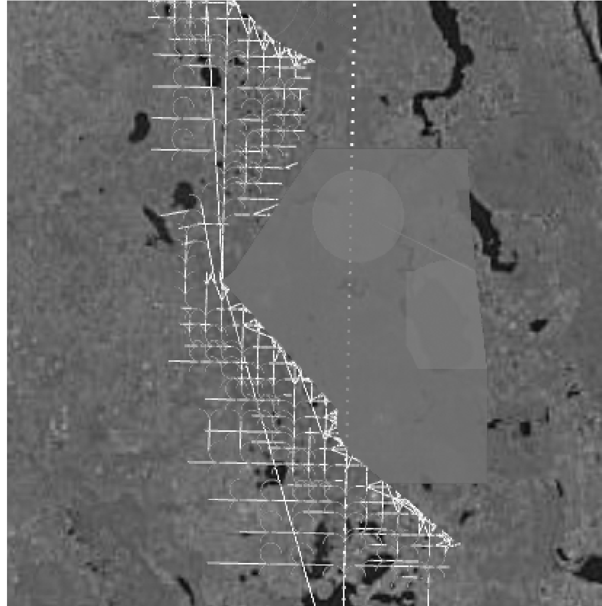

Fig. 5 Part of the state space for the scenario R1. The object on the right hand side is the Special Use Airspace. The generated states are on the left hand side. The original airplane trajectory connecting the start position and goal position directly is dotted.

6 Conclusion

In the paper, the parallelization extension of the AA* algorithm is studied. This extension is called PAA* algorithm and combines the ideas from the HDA* algorithm and the AA* algorithm. PAA* runs on multi-core/multi-processor computers and utilizes the asynchronous messaging and distributed open and closed lists. The experiments proved that the speed-up of PAA* is remarkable, yet dependent on the selected hash function (partitioning of the geographical search space) and on the configuration of the environment. PAA* is able to provide an overall speed-up of 2.15 for 8 cores for the artificial scenario and 1.39 for 8 cores for the real-world scenario even using the simple hashing function. In our opinion, an interesting topic for the future work is the hash function selection. Depending on the domain and the used optimization criterion for the A*, it makes sense to investigate the way the hash function assignes partitions of the 2-D universe to the cores. This is an optimization problem, where criterions are the utilization of cores and also the number of the nodes that are sent to a different cores.

7 Acknowledgements

The research presented in this paper has been supported by the Czech Technical University SGS grant No SGS10/191/OHK3/2T/13. The Accelerated A* algorithm has been supported by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-06-1-3073 and by Czech Ministry of Education grant number 6840770038. The views and conclusions contained herein are those of the author and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the U.S. Government or the Czech Government.

References

1. Ethan Burns, Seth Lemons, Rong Zhou, and Wheeler Ruml. Best-first heuristic search for multi-core machines. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 449–455, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
2. Ethan Burns, Sofia Lemons, Rong Zhou, and Wheeler Ruml. Parallel best-first search: The role of abstraction. In *Proceedings of the AAAI-10 Workshop on Abstraction, Reformulation, and Approximation*, 2010.
3. Matthew Evett, Ambuj Mahanti, Dana Nau, James Hendler, and James Hendler. Pra*: Massively parallel heuristic search. *Journal of Parallel and Distributed Computing*, 25:133–143, 1995.
4. A. Kishimoto, A. Fukunaga, and A. Botea. Scalable, Parallel Best-First Search for Optimal Sequential Planning. In *Proceedings of the International Conference on Automated Scheduling and Planning ICAPS-09*, pages 201–208, Thessaloniki, Greece, 2009.
5. Michal Pěchouček and David Šišlák. Agent-based approach to free-flight planning, control, and simulation. *IEEE Intelligent Systems*, 24(1), January-February 2009.
6. David Šišlák, Přemysl Volf, and Michal Pěchouček. Accelerated A* trajectory planning: Grid-based path planning comparison. In *Proceedings of the 19th International Conference on Automated Planning & Scheduling (ICAPS)*, pages 74–81, Menlo Park, California, 2009. AAAI Press.
7. David Šišlák, Přemysl Volf, and Michal Pěchouček. Flight trajectory path planning. In *Proceedings of the 19th International Conference on Automated Planning & Scheduling (ICAPS)*, pages 76–83, Menlo Park, California, 2009. AAAI Press.
8. David Šišlák, Přemysl Volf, and Michal Pěchouček. Accelerated A* path planning. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, New York, May 2009. ACM Press.