An Operator Distribution Method for Parallel Planning

Dimitris Vrakas, Ioannis Refanidis & Ioannis Vlahavas

Department of Informatics,
Aristotle University of Thessaloniki
54006, Thessaloniki Greece
[dvrakas,yrefanid,vlahavas]@csd.auth.gr

Abstract

This paper presents the ***O**perator **D**istribution **M**ethod for Parallel **P**lanning* (ODMP), a parallelization method especially suitable for heuristic planners. ODMP distributes the process of finding and applying the ground applicable actions to a given state, to the set of the available processors. The operator schemas of the domain are distributed to the available processors in a dynamic manner. In order to utilize a larger number of processors and to achieve better load balancing, the set of the domain's operators is initially expanded by considering all the possible instantiations of their first argument. The proposed method, ODMP, is an effective parallelization method for heuristic planners, but it can also be applied to planners that embody other search strategies as well. We implemented ODMP in a best first planner that uses a domain specific heuristic for logistics problems and tested its efficiency on a variety of problems, adopted from the AIPS-98 planning competition.

1 Introduction

Heuristic functions are an important component of many artificial intelligence applications, especially when a "quite good" (not necessarily optimal) solution is required and there is a tight time limit. Planners are Artificial Intelligence applications, which given an initial state *I*, a set of possible actions *A* and certain goals *G*, produce a plan of actions, which if applied to *I* achieves *G*. These programs are in many cases embedded in systems that must exhibit real-time behavior, so they are usually equipped with heuristic functions in order to respond promptly. Speed is one of the most desirable aspect of planning systems and although various methods, like hierarchical planning, case based planning, transformation to other problem types e.t.c., have been adopted, the absence of good heuristic functions makes planning systems inefficient for practical domains.

A challenging feature of modern artificial intelligence applications is the ability to distribute the workload among several processors in order to increase the execution speed. Although the technology of parallel architectures is quite mature and a lot of parallel systems are available at a reasonable cost, there are not many software products that can utilize these possibilities. Many researchers have tried to find parallelization techniques for AI applications and they have mainly focused on ways to distribute the search tree among the existing processors (Cook and Varnell 1999, Cook 1997, Kumar and Rao 1990, Powley and Korf 1991, Rao, Kumar and Ramesh 1987). These techniques, enriched with load balancing (Kumar, Grama and Rao 1994) and operator reordering methods (Cook, Hall and Thomas 1993, Powley and Korf 1991), produce quite efficient parallel algorithms. However the efficiency of these methods depends on the amount of redundant search that will be inevitably performed and therefore are not suited for heuristic planning.

This paper presents the ***O**perator **D**istribution **M**ethod for Parallel **P**lanning* (ODMP), a parallelization method especially suitable for efficient heuristic planners in the STRIPS framework (Fikes and Nilson 1971). In ODMP, which is based on our previous work on the parallelization of GRT (Vrakas et al. 1999), the task of finding and applying the applicable ground actions to the current state is done in parallel. Furthermore, in order to utilize a larger number of processors and to achieve better load balancing, the set of operators is initially expanded by considering all the semi-grounded operators that can be generated by the possible instantiations of the operators' first arguments. The expanded set is then distributed, in a dynamic manner, to the available processors.

ODMP is an effective parallelization method for heuristic planning, since any other parallelization method introduces redundant search, which is so more severe, as the heuristic becomes more precise. However it can co-operate with other optimization techniques and it can offer additional speedup to other kinds of planning.

In order to measure the efficiency of ODMP, we developed a simple best-first planner with a domain-dependent heuristic for the logistics world (Veloso 1992). We measured the performance of a sequential version of the planner against a parallel version that uses the ODMP approach, obtaining very promising results.

The rest of the paper is organized as follows: Section 2 gives a brief synopsis of the work related to planning and parallel search algorithms. Section 3 introduces a method that expands the operator set by considering the semi-grounded operators. This approach enables our parallelization method to utilize a large number of processors. Section 4 describes ODMP, our proposed parallelization method, while section 5 describes the heuristic for logistic problems we implemented for our experiments. Section 6 presents some experimental results on a variety of logistics problems and finally section 7 concludes the paper and poses future directions.

2 Related Work

In (Kumar, Rao and Ramesh 1988), the authors review a set of strategies for parallel best-first search of state-space graphs. The strategies they present are classified to be

either distributed or centralized, based on the existence or not of local agendas. In both cases the heuristic function is used to order the states in the agenda, i.e. the first state in the agenda is the one with the smallest estimated distance from a goal state.

In the centralized model, each one of the *N* processors undertakes the best state of the global agenda, which has not yet been assigned to any other processor. At the end of each expansion the successor states are placed back to the global agenda. The main advantage of this approach, as discussed in (Irani and Shih 1986), is that it does not result in much redundant search. However, the global agenda is accessed by all the processors very frequently causing the processors to remain idle for quite a long time, due to contention.

On the other hand, in the distributed model each processor maintains its own local agenda and thus there is no need for semaphores. This model usually uses the IDA* search algorithm initially presented by Powley and Korf (Powley, Ferguson and Korf 1991). IDA* is a version of Iterative Deepening search, where the next level of search is determined by the heuristic function in use. The state-space is initially divided and distributed to the existing processors. The segmentation of the initial state-space can be done in several ways. Powley and Korf (Powley and Korf 1991) introduced PWS, a tree distribution method in which each processor searches in a unique depth. Kumar et al. (Kumar and Rao 1990, Rao, Kumar and Ramesh 1987) describe a different approach where the search tree is segmented vertically. To be more specific, after a sufficient number of states has been generated, each processor undertakes one of them, considering it to be the root and searches the generated subtree. A large number of variations of these techniques have been proposed over time. Moreover, Diane Cook proposed a hybrid approach (Cook and Varnell 1999, Cook 1997), which combines IDA* with vertical segmentation techniques and seems to outperform all the other methods.

In the above method, after the initial distribution of the state-space, some intercommunication is necessary, since some of the processors may be working on promising parts of the search tree while the others contribute little or nothing to the process of finding a solution. Moreover, the communication is necessary for load balancing, since the local agenda of a processor may become empty if many non-expandable states have been examined (Kumar, Grama and Rao 1994). Load balancing includes the transfer of states from one local agenda to another, in order to equalize the workload in all processors. This transfer can be performed directly or via a global memory structure, called blackboard.

There are two main problems related with the kind of parallelization based on the distribution of the search space: a) a great number of states is examined more than once, since the state-space is not always split in disjoined parts and b) these techniques result in the expansion of more states than necessary. The first argument does not apply to IDA* since the search tree is split in almost disjoined parts, except for the states that can be approached by various ways of different length. However, IDA* examines all the states at a given level before proceeding to the next one (argument b). The alternative approach (vertical segmentation) suffers from both problems. The subtrees can not be disjoined, since a state can usually be approached by different ways. Furthermore, a subtree might be promising (i.e. it contains a short solution), while the others are not and yet the algorithm will examine all of them.

The latter problem becomes more severe as the heuristic function produces better estimates, since the set of promising states will become narrower and narrower. For example, if the heuristic function is perfect, a simple hill climbing technique will examine only *l* states, where *l* is the length of the optimal solution. Any one of the parallelization methods described previously will work *N* (number of processors) times more, since while one of the processors will be examining the solution's states the others will be wasted at useless parts of the search space. Even if the accuracy of the heuristic estimate is less than 100%, but still acceptable, the overhead imposed by the examination of redundant states would not allow the parallel algorithm to perform well.

These inefficiencies were partly tackled by PGRT (Vrakas et al. 1999), a parallelization method for planning especially crafted for *Greedy Regression Tables* (Refanidis and Vlahavas 1999, Refanidis and Vlahavas 2000), a domain independent heuristic for planning. In PGRT the operators are dynamically distributed to the available processors and the task of formatting the successor states is done in parallel. PGRT manages to achieve significant speedup for a small number of processors (3 or 4 for the logistics domain) while preserving the quality of the resulting plans. The main disadvantage of PGRT concerns its scalability, since the number of operators in most domains is relatively small.

3 Expanding the Operator Set

ODMP distributes the task of finding and applying the applicable grounded actions to a given state. It is therefore obvious that the number of operators limits its scalability. Furthermore, there are great differences among the amount of work needed to examine these operators and inevitably the workload among the processors will not be balanced. PGRT managed to cope, partially, with the latter problem through the use of an *Operator Reordering* method as a means of load-balancing. ODMP has adopted a more sophisticated method, which can cope efficiently with both of the problems stated above.

The set of operators is initially expanded through the consideration of all the possible instantiations of the operators' first argument. We call the resultant operators *semi-grounded* operators and the expanded set *semi-grounded operator set*.

Consider for example, a logistics domain with 3 cities (`city1`, `city2` and `city3`), 1 plane (`A321`), 2 places per city (`center`, `airport`), 3 trucks (`truck1`, `truck2` and

truck3) and 4 cargoes (cargo1, cargo2, cargo3 and cargo4). The initial operator set includes the following six operators:

[*Fly(A,S,D), Drive(T,S,D), Load_plane(C,A,L), Unload_plane(C,A,L), Load_truck(C,T,L), Unload_truck(C,T,L)*]

where A,C,D,L,S and T are variable names representing airplanes, cargoes e.t.c. The semi-grounded operator set will contain one instantiation of the *fly* operator, three of the *drive* operator and four of each one of the other operators. So the size of the expanded set will be twenty. For example, the three semi-grounded *Drive* operators will be the following:

[*Drive(truck1,S,D), Drive(truck2,S,D), Drive(truck3,S,D)*]

The semi-grounded operator set contains a larger number of operators than the initial set, but the amount of work needed to process these sets is the same, since the number of actions that will eventually be generated by these sets will be equal. So, the method is capable of generating a relatively large number of disjoined segments of operators, which is equivalent to the initial operator set.

The efficiency of the parallel algorithm can benefit from this method in several ways:

a) The algorithm can utilize the computational power of more processors (the bound in the scalability is lifted).
b) Since the workload is split into a larger number of portions, it's distribution will be more balanced.

The choice of the argument, which will be used to generate the semi-grounded operators can significantly affect the efficiency of ODMP. In the current implementation we always choose the first argument appearing in the definition of the operator.

4 Overview of ODMP

ODMP is a method for parallel planning, in which the task of finding and applying the applicable actions to a given state is done in parallel. To be more specific, suppose that we have *M* operators and *N* processors. We distribute the operators to the available processors and then each processor is responsible of finding the applicable ground actions originating from the operators assigned to it and applying them to the current state to produce the successor states. At the beginning of the planning process ODMP expands the operator set using the method described in section 3. The rest of the planning process uses the set of the *semi-grounded operators* instead.

The distribution could be done statically at the beginning; i.e. the first $\lceil M/N \rceil$ operators will be assigned to the first processor, the next $\lceil M/N \rceil$ operators to the second processor and so on. This approach is easy to implement and the overhead due to the communication among processors is kept quite low. However, the number of ground actions originating from different operators can vary from 0 to several hundreds (for a typical logistics problem) resulting in unbalanced workload among the different processors.

In the dynamic distribution method the unexamined operators are kept all together in a global data structure, denoted as *operator pool*. Initially each processor is assigned one operator and the rest of the operators are distributed on demand. This method succeeds to balance the workload among processors, but imposes some overhead due to contention. However, this overhead is negligible compared to the speedup due to the balanced workload.

As it was previously stated, ODMP is a parallelization method that can be adopted by any state-space planner. However, it was motivated by the need for an effective parallelization method for heuristic planning. Figure 1 presents the algorithm of ODMP for best-first planning. It is obvious that the steps presented in Figure 1 do not form a complete description of the algorithm. This is just the part of the parallel algorithm running on the different threads, which will generate and apply the applicable actions to the current best state S_B. There must also exist a controlling process that will be responsible for starting/stopping the threads and for assigning S_B to the current most promising state.

1. While S_B has not been defined, do nothing.
2. While operator pool is not empty:
   - 2a. Request an operator
   - 2b. Find all the grounded actions that can be applied to S_B.
   - 2c. Send the list of grounded actions to the action pool.
3. While action pool is not empty or there is at least one processor at step 2:
   - 3a. Request new action.
   - 3b. Apply it to S_B to produce S'.
   - 3c. Evaluate the distance of S' from the goal state using the heuristic function.
   - 3d. Send (S',dist(S')) to the global agenda.
4. Return to 1.

Figure 1. ODMP for heuristic planning

The first step of the parallel algorithm is used for synchronization between the various processors. The value of S_B will be updated only when all the processors have finished with the previous iteration. This part is crucial, since if a processor was allowed to start a new iteration while the others are still working with the previous one, S_B would be linked to a local best state that probably wouldn't be the globally best one according to the heuristic in use. The last one would have resulted in greater CPU usage, but also in larger number of examined states and consequently larger execution time.

In order to achieve further increase in the efficiency of the parallelization, the grounded applicable operators are temporarily stored in another pool, denoted as *action pool* and the remaining tasks, i.e. creation of successor states and evaluation using the heuristic function, are done in an independent phase (step 3). This technique can offer further increase in CPU usage, since it contributes to better load balancing.

In order to apply ODMP to a non-heuristic planner, the algorithm in Figure 1 has to be modified. Step 3c is unnecessary since there is no heuristic algorithm in use. Furthermore, the global agenda will not contain tuples of the form (S,dist(S)), since the estimated distances are not available. Finally, there is no need to have a separate step for the creation of successor states, since without the use of a heuristic algorithm this task is trivial.

5 The Cargo Location Heuristic

In order to test the efficiency of ODMP, we embodied it in a simple best-first planner that uses a domain-dependent heuristic algorithm for Logistics worlds. We call this heuristic the *Cargo Location* (CL) heuristic. CL estimates the distances between each intermediate state and the goals, taking into consideration the current locations and the destinations of the cargoes that have to be transferred.

To be more specific, for each cargo *c*, CL assigns an integer varying from 0 to 12, which represents the estimated number of steps for this cargo to reach its destination (d_c). Then, the sum of these distances is the estimate for the distance of each intermediate state from the goals. Initially each d_c is set to 0. Then the d_cs are computed by repeatedly applying for each cargo the following rules:

1. If *c* is not in its destination city, increase d_c by 4.
2. If c is not in its destination city and it is not in an airport, increase d_c by 4.
3. If c is not in its destination city and its destination place is not an airport, increase d_c by 4.
4. If c is in its destination city but it is not in the destination place, increase d_c by 4.

Consider, for example, the following case:

```
Goals≡   [at(c1,dc-ctr), at(c2,la-air),
          at(c3,la-ctr)]
StateA≡  [at(c1,dc-air), at(c2,dc-air),
          at(c3,dc-ctr)]
```

The distance between `StateA` and the `Goals` is estimated as follows:

Estimated distance between `StateA` and `Goals`
`cargo1`: 4 (4th rule)
`cargo2`: 4 (1st rule)
`cargo3`: 12 (1st, 2nd and 3rd rule)
Total = **20**

6 Performance Results

For evaluation reasons, we developed a planning system that embodies ODMP. Our planner searches the state-space in a best-first manner and uses the CL heuristic to guide its search. The implementation language was C++ with multithreading capabilities (we used the POSIX threads library). The platform used for the tests was a SGI Power Challenge XL parallel machine with 14 R8000 CPUs (75 MHz) and 16 GB of shared memory. The underlying operating system was IRIX 6.2.

We tested the efficiency of ODMP on the thirty Logistics problems used in the AIPS–98 planning competition (prob01 – prob30). However in the following Tables we only consider a subset of them, which required a reasonable amount of processing time. The ten selected logistics problems were thoroughly used as inputs for various versions of our planner, utilizing each time a different number of processors in the range between 1 and 12.

Table 1 presents the time needed by our planer to solve a variety of hard logistics problems for different values of *N* (the number of utilized processors). The time was measured using the Unix's *time* command (User time + Sys time) and it is presented in seconds.

Problem	*N*=1	*N*=2	*N*=3	*N*=5	*N*=8	*N*=12
Prob09	254	144	110	70	52	55
Prob10	512	275	207	150	120.5	108
Prob12	2100	1200	900	600	450	370
Prob13	2660	1385	946	630	420	320
Prob14	265	152	120	80	57	49
Prob16	224	128	101	67	49	44.2
Prob17	118	72	50	33	28.1	33.7
Prob18	3900	2046	1415	855	570	450
Prob19	1350	800	523	366	270	220
Prob23	55	33	24	20	17	18

Table 1. Time needed by ODMP for different values of *N*

In order to illustrate the efficiency and the scalability of our method in more clarity, we computed the *speedup* ($\mathbf{T}_{sequential}/\mathbf{T}_{parallel}$) of ODMP from the values of Table 1. The, quite interesting, results are presented in Table 2.

Problem	*N*=1	*N*=2	*N*=3	*N*=5	*N*=8	*N*=12
Prob09	1	1.76	2.31	3.63	4.88	4.62
Prob10	1	1.86	2.47	3.41	4.49	4.74
Prob12	1	1.75	2.33	3.5	4.67	5.68
Prob13	1	1.92	2.81	4.22	6.33	8.31
Prob14	1	1.74	2.21	3.31	4.65	5.41
Prob16	1	1.75	2.22	3.34	4.57	5.07
Prob17	1	1.64	2.36	3.58	4.2	3.5
Prob18	1	1.91	2.76	4.56	6.84	8.67
Prob19	1	1.69	2.58	3.69	5	6.14
Prob23	1	1.67	2.29	2.75	3.24	3.06

Table 2. Speedup of ODMP

By analyzing the results presented in the preceding Tables we can draw certain important conclusions:

- Under certain circumstances ODMP can achieve almost-linear speedup (e.g. speedup for Prob13 and *N*=3 is 2.81)
- It can effectively scale up to a significant number of processors (approximately 10 in the case of Prob18).
- The efficiency of our method seems to depend strongly on the nature of the problem, since for certain problems (Prob17 and Prob23) the speedup and the scalability of ODMP are moderate.

In order to justify the behavior of our method, we performed an analysis on the set of the logistics problems regarding their inner structure and its impact on our method. For each problem we computed the size of the *semi-grounded operator set* (denoted as SGOS) and the upper limit of the number of grounded actions that can be applied to a given state (A_{max}). Table 3 presents the results of our analysis.

Problem	SGOS	A_{max}
Prob09	80	204
Prob10	95	145
Prob12	84	409
Prob13	130	385
Prob14	171	185
Prob16	93	248
Prob17	110	135
Prob18	120	390
Prob19	117	318
Prob23	116	98

Table 3. Number of semi-grounded operators and applicable actions

Comparing the values of A_{max} (Table 3) and the *speedup* (Table 2), we can conclude that the efficiency of ODMP depends strongly on A_{max}. This sounds quite reasonable, since the most resource-consuming part of the planning process is the detection of the applicable actions and the formation of the successor states. The scalability of ODMP and therefore its overall efficiency, is also affected by the number of semi grounded operators (SGOS), since a low value for SGOS, means that there are not many work packages and the workload is not equally distributed. For example, although A_{max} for Prob12 is very high (409), the quite low value of SGOS (84), prevents ODMP from performing well.

7 Conclusion and Future Work

This paper reported on work performed to find an adaptive parallelization method for planning. We proposed a method that distributes the process of finding and applying the grounded applicable actions to a given state. The Operator Distribution Method for parallel Planning (ODMP) was inspired by the parallelization method of PGRT [15]. The main disadvantage of PGRT was the limitation in its scalability, which was tackled successfully by ODMP, through the creation of the Semi-Grounded Operator Set. This expanded set is created through the consideration of all the grounded instantiations of the operators' first argument.

ODMP is a parallelization method that can be adopted by any planner, whether it embodies a heuristic function or not. For the purpose of this research we implemented a best first planner using a simple heuristic for logistics problems (CL). We tested this program thoroughly on a large variety of logistics problems and the results presented in this paper illustrate the efficiency of ODMP.

In the future we plan to apply ODMP to other heuristic planners and test the efficiency of our approach, thoroughly, on other, probably more complicated domains.

We also plan to study the possibility of applying ODMP to parallel machines with distributed memory and afterwards in a network of computers. This will require a static work distribution method in order to minimize the intercommunication, which could be probably done through a heuristic function capable of estimating, a priori, the workload imposed by an operator.

8 References

Cook, D. J. and Varnell, R. C. 1999. Adaptive Parallel Iterative Deepening Search. Journal of Artificial Intelligence Research 9: 167-194.

Cook, D. J. eds. 1997. A Hybrid Approach to Improving the Performance of Parallel Search. Parallel Processing for Artificial Intelligence.: Elsevier Science Publishers.

Cook, D. J., Hall, L. and Thomas, W. 1993. Parallel search using transformation-ordering iterative-deepening A*. The Int. Journal of Intelligent Systems. 8(8).

Fikes, R. E. and Nilsson, N. J 1971. Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2.: 189-208

Irani, K. B. and Shih, Y. F. 1986. Parallel a* and ao* algorithms: An optimality criterion and performance evaluation. In Proceedings of the International Conference on Parallel Processing, 274-277.

Kumar, V., Rao, V. N. and Ramesh, K. 1988. Parallel Best-First Search of State-Space Graphs: A Summary of Results. In Proceedings of the 1988 National Conf. on Artificial Intelligence.

Kumar, V., Grama, A. Y. and Rao, V. N. 1994. Scalable Load Balancing Techniques for Parallel Computers. Journal of Parallel and Distributed Computing 22: 60-79.

Kumar, V. and Rao, V. N. eds. 1990. Scalable parallel formulations of depth-first search. Parallel Algorithms for Machine Intelligence and Vision.: Springer-Verlag.

Powley, C., Ferguson. C. and Korf, R. E. 1991. Parallel tree search on a simd machine. In Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, 249-256.

Powley, C. and Korf, R. E. 1991. Single-agent parallel window search, IEEE Transactions on Pattern Analysis and Machine Intelligence 13(5).

Rao, V. N., Kumar, V. and Ramesh, K. 1987. A parallel implementation of iterative deepening-A*. In Proceedings of the National Conference on Artificial Intelligence, 178-182.

Refanidis, I. and Vlahavas, I. 1999. *GRT: A Domain Independent Heuristic for STRIPS Worlds based on Greedy Regression Tables.* In Proceedings of the 5th European Conference on Planning. Durham, UK.

Refanidis, I. and Vlahavas, I. 2000. Exploiting State Constraints in Heuristic State-Space Planning. Forthcoming.

Veloso, M. 1992. Learning by Analogical Reasoning in General Problem Solving. Ph.D. diss., Dept. of Computer Science, Carnegie Mellon Univ.

Vrakas, D. Refanidis, I., Milcent, F. and Vlahavas, I. 1999. On the Parallelization of Greedy Regression Tables. In Proceedings of the 18th Workshop of the UK Planning and Scheduling Special Interest Group, 180-189. Manchester UK.