A BLACKBOARD SYSTEM FOR DISTRIBUTED ANALYSIS OF IMAGE SEQUENCES

K. Welz[1], C.-E. Liedtke[1]

Abstract

The analysis of image sequences, which have been obtained by a TV camera from a natural scene, requires in general a large number of different tasks. Under real-time conditions many of these tasks have to be executed simultaneously. This paper proposes a multiprocessor architecture, which supports the system requirements for such kind of image sequence analysis problems. Interprocess communication and task distribution are realized with the blackboard principle. To study the system requirements of a multiple task analysis system we have chosen, as an example, the automated recognition of traffic signs from a moving car.

1. Introduction

Execution time for the analysis of natural image sequences depends on the complexity of the scene. The more relevant objects the scene includes, the longer is the time needed for execution. Therefore, the analysis of natural image sequences using sequential task execution is often not possible, when real-time operation is requested. Parallel processing is necessary to increase processing power and system reliability. Increasing time requirements of an application problem should be compensated by a modular system extension, without changing any application software. For this reason, the whole problem must be divided into small independent subtasks, which can be distributed to the available processors /1/. Parallel task execution requires the exchange and collection of data and execution results. Cooperation between the parallel working processors is needed in order to increase reliability, availability, and performance of the whole system /2/. Because of the uncertainty in interpreting natural image sequences, a pure top down or bottom up strategy is not possible. The required mechanisms have to enable cooperation within and between all levels of image analysis, i.e. the numeric and the symbolic processing level. Type and number of tasks waiting to be executed at the same time depend on the content of the actual scene. These tasks have to be allocated to the processors /2, 3, 4/. For application problems, where the types and numbers of tasks vary during execution, an automatic distribution of the tasks to the available processors will be more efficient than static allocation, because it more effectively balances computation and communication. By a dynamic distribution of processing power image processing tasks can be done with a smaller set of processors.

In particular applications, like the navigation of vehicles in natural environments, the real time performance can be the most important criterion. Efficiency and cost effectiveness is important, but is not in all cases of utmost concern. The primary intention of this project is to develop an architecture, which permits real-time operation for a large range of image sequence analysis problems, based on multiprocessing. It is not the primary purpose to maximize processor efficiency.

[1] Institut für Theoretische Nachrichtentechnik und Informationsverarbeitung, Universität Hannover, Appelstr. 9A, D-3000 Hannover 1, FRG

2. A Blackboard for Parallel Image Analysis

2.1 Blackboard Principle

In a top down approach to sequence analysis, which is envisioned here, tasks have to be generated during run time. These tasks must be allocated dynamically to the free processors. An advantage of an automatic distribution strategy is a high flexibility in case of system extensions.
For task distribution we will use the blackboard principle which provides information about executable tasks and processing results. Processors are able to write new tasks and results onto the blackboard. Idling processors can read a new executable task from the blackboard. Processing results will be distributed via the blackboard.

2.2 Blackboard Structure

A blackboard entry specifies a task. Each task has a unique *number*. Tasks may have different types of *destination*. A task with the destination attribute "broadcast" has to be executed by each processor. The destination attribute "broadcast" is suitable for the distribution of data, which are important for each processor. Having the destination attribute "any" a task may be executed by any processor. The destination attribute "definite" specifies that a task has to be executed on a definite processor or a definite set of processors.

Each task has a *priority*. A high priority guaranties that very important tasks or very complex tasks will be processed first. Answer tasks will get a high priority, too, so that processing results will be returned as quickly as possible. The blackboard is organized as a priority sorted waiting queue. When creating a new blackboard entry, the creation *time* will be noticed. Priority will increase with increasing waiting time. A dynamic priority avoids that tasks with low priority will never be executed, if there are tasks with a higher priority generated permanently.

To differentiate between executable tasks, waiting tasks, and tasks containing processing results a blackboard entry includes a *status* field. The *type* of a task specifies the algorithms to be executed. To achieve a good computation balance each processor should be able to execute as many different tasks as possible. A task entry has input or output parameters. The *parameter* field contains the length of the data block and the data itself.

2.3 Blackboard Realization

In Shared Memory Systems a blackboard can be realized as a globally accessible data base. All medium- and high-level processors have access to the blackboard. A disadvantage is the fact that a master processor is needed for blackboard administration. Access to the blackboard may become a bottleneck. For large systems this central distribution strategy is not efficient.

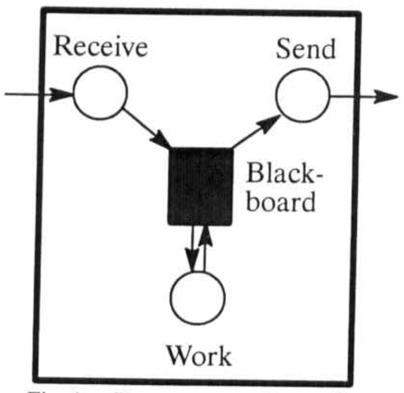


Fig. 1. Processes on a Dynamic Blackboard Processor

Since we are mainly interested in large multiprocessor systems (not supercomputers), to avoid a system bottleneck only decentral schemes are considered. In loosely coupled systems with message passing we have the possibility to realize the blackboard principle for the data exchange as a dynamic blackboard. In a dynamic blackboard architecture blackboard entries will be distributed via the whole system. Each processor administers it's own local blackboard, which is organized as a priority sorted waiting queue. For this reason, at any time, each individual processor has access to a small part of the blackboard, only. Blackboard entries will be exchanged between the processors using special distribution strategies.

Fig. 1 shows that on each processor participating in the dynamic blackboard setup, three processes are running. All processes have access to a waiting queue, which contains a small part of the blackboard. The receive process receives tasks and writes them into the waiting queue. The work process may read a task from the waiting queue, executes this task, and writes processing results and/or new tasks into the waiting queue. The send process sends tasks, which have to be executed by other processors, and results to the next processor.

3 Results

In our present experimental setup the dynamic blackboard is realized on a transputer network, employing the transputer links for data exchange. First tests regarding parallel execution of image processing tasks are made. These tests measure speed up and overhead for a parallel task execution on the dynamic blackboard system. Fig. 2 shows the basic structure of our system, generally consisting of several rings of dynamic blackboard processors. To study the system requirements of the dynamic blackboard system we use the automated recognition of traffic signs from a moving vehicle. For these first tests we designed a symmetrical organized dynamic blackboard. In this example we have only one blackboard (processor ring). The strategy of distributing tasks over the system is to send tasks to the next processors blackboard, when a processor is busy. Compared to other distribution strategies, where idling processors ask other processors for a task, our approach avoids additional overhead of idling processors.

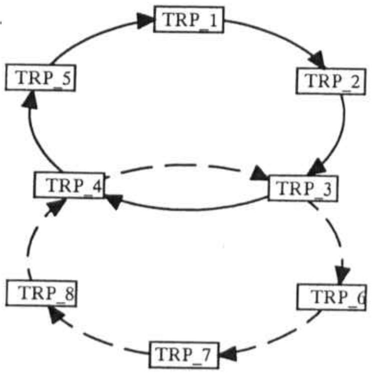


Fig. 2. Dynamic Blackboard System consisting of two Processor Rings

The two conventional methods for parallel operation of image processing applications are data partitioning and functional partitioning. Data partitioning in image sequence analysis means that several processors execute the same task on different parts of the image. Functional par–

titioning means that different tasks are executed on different processors. The tasks which have to be executed in parallel in connection with the traffic sign recognition are for instance observing the left and right road edge, searching for new candidates for traffic signs along the road edge, and tracking /5/ as well as interpreting possible traffic signs. Fig. 3 illustrates the types of parallel execution in our system.

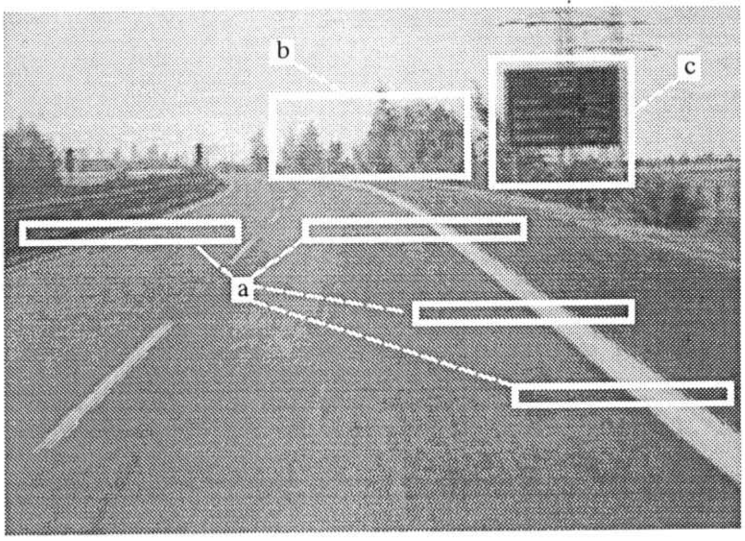


Fig. 3. Road Scene with Windows for Road Edge Tracking (a), Candidate Search (b), and Traffic Sign Tracking and Interpreting (c)

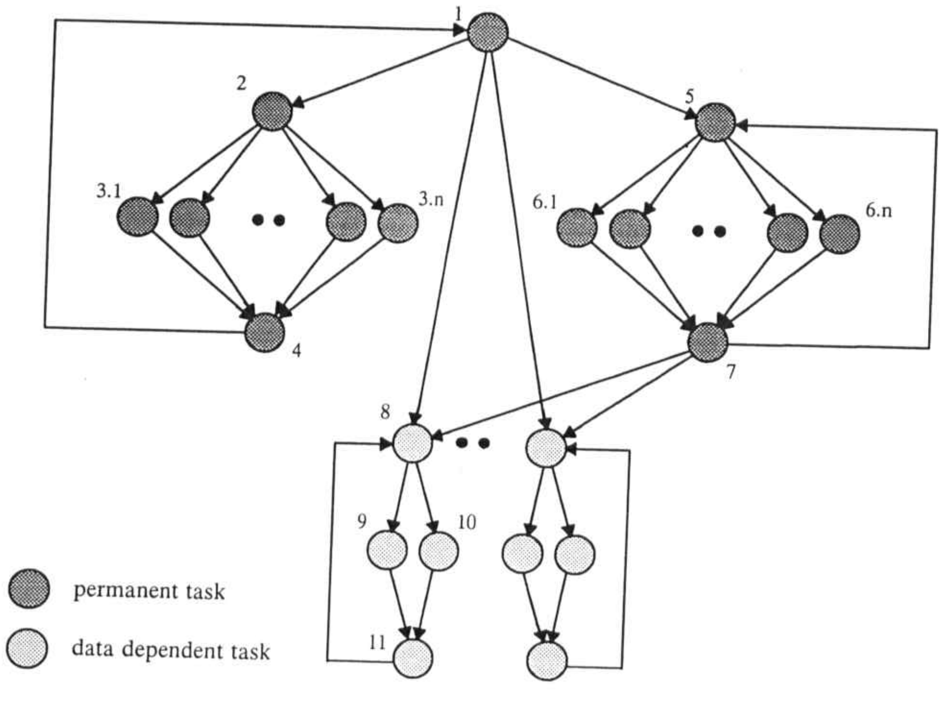
The strategy in finding road edges is the subdivision of the problem into an initial global analysis of the roadway and the tracking of the road edges. For the global analysis region orientated segmentation methods are used /6, 7/. When knowing the location of the road edges, it is sufficient to determine changes in the location in small control windows /8/, because of the little changes between two images. For this reason, tracking road edges can be executed in parallel for different locations within the TV image plane by different processors. The tracking of the road edges by several windows represents data partitioning. The results supplied by the parallel road edge tracking will be combined in a road model. The model knowledge controls the positioning of the windows for road edge tracking and candidate search.

A window for searching candidates for traffic signs can be set in that area where new traffic signs appear with a high likelihood. For the purpose of searching candidates for traffic signs we use different methods i.e. searching for triangles, circles, or areas of specific colors. This is an example for functional partitioning, because there are different tasks for the same window.

Having found a distant candidate for a traffic sign it cannot be identified, because of its small size. Hypotheses about the type of the detected traffic sign will be generated. These hypotheses will be verified in each new image. A hypothesis will be confirmed, rejected, or step by step refined. The smaller the distance to the traffic sign, the more information about the traffic sign is available, so that the hypothesis can be more specific. For the purpose of observing a traffic sign it must be tracked.

Fig. 4 shows the task graph for the automated recognition of traffic signs from a moving car. For each image the tasks 1 to 7 have to be executed. The tasks 8 to 11 have to be executed only, if a candidate for a traffic sign has been detected. These tasks are data dependent. The type and number of parallel working tasks is dynamically changing.

The execution time for the parallel execution of the tasks from the task graph from Fig. 4 in dependence on the number of working tranputers participating in the dynamic blackboard set up is measured. The test varies the number of working transputers from 1 to 5. In this example we have 8 tasks for "road_edge_tracking", 4 different tasks for "candidate_search", and two candidates for traffic signs, which have to be tracked and interpreted. Fig. 5 shows the execution time and Tab. 1 the speed up depending on the number of transputers.



Task 1:	init_road_model
Task 2:	init_road_edge_tracking
Task 3:	road_edge_tracking
Task 4:	update_road_model
Task 5:	init_candidate_search
Task 6:	candidate_search:
	- searching_for_circles
	- searching_for_triangles
	- searching_for_colored_areas
Task 7:	init_candidate_list
Task 8:	init_traffic_sign_model
Task 9:	traffic_sign_tracking
Task 10:	traffic_sign_feature_extraction
Task 11:	traffic_sign_interpretation

Fig. 4. Task Graph for Traffic Sign Recognition

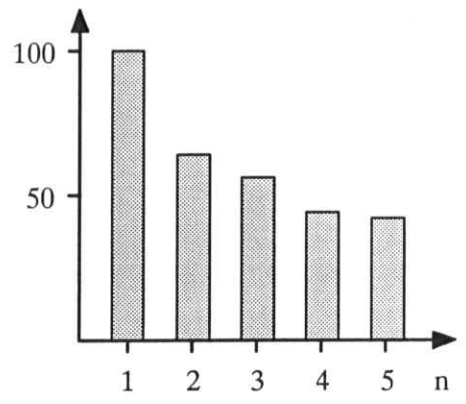


Fig.. 5. Relative Execution Time Depending on the Number of Transputers n

Number of Transputers n	Speed Up
1	1.00
2	1.54
3	1.79
4	2.26
5	2.38

Tab. 1. Speed Up Depending on the Number of Transputers n

4. Acknowledgements

The work has been supported by the Deutsche Forschungsgemeinschaft (DFG).

5. References

[1] Howe, Carl D., Bruce Moxon: "How to program parallel processors", IEEE Spectrum, September 1987.

[2] Ni Lionel M., Chong-Wei Xu, Thomas B. Gendereau: "A Distributed Drafting Algorithm for Load Balancing", IEEE Trans. on Software Engineering, Vol. SE-11, No. 10, October 1985.

[3] Lin Frank C. H., Robert M. Keller: "The Gradient Model Load Balancing Method", IEEE Trans. on Software Engineering, Vol. SE-13, No. 1, January 1987.

[4] Chou Timothy C. K., Jacob A. Abraham: "Load Balancing in Distributed Systems", IEEE Trans. on Software Engineering, Vol. SE-8, No. 4, July 1982.

[5] Liedtke C.-E., H. Busch, R. Koch: "Automatic Modelling of 3D Moving Objects from a TV Image Sequence", SPIE/SPSE Symposium on Electronic Imaging, Santa Clara, CA., USA, Feb. 1990.

[6] Turk Matthew A., David G. Morgenthaler, Keith D. Gremban, Martin Marra: "VITS-A Vision System for Autonomous Land Vehicle Navigation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 3, May 1988.

[7] Thorpe C., M. H. Hebert, T. Kanade, S. A. Shafer: "Vision and Navigation for the Carnegie-Mellon Navlab", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 3, May 1988.

[8] Dickmanns E. D., Th. Christians: "Relative 3D-State Estimation for Autonomous Visual Guidance of Road Vehicles", Intelligent Autonomous Systems 2 (IAS-2) Amsterdam, 11-14 December 1989.