11-1988

Parallel Split-Level Relaxation

Thomas C. Henderson
University of Utah, tch@cs.utah.edu

Ashok K. Samal
University of Nebraska-Lincoln, asamal1@unl.edu

Henderson & Samal in Proceedings of the Ninth International Conference on Pattern Recognition (1988)

Parallel Split-Level Relaxation[1]

Thomas C. Henderson and Ashok Samal

Department of Computer Science
University of Utah, Salt Lake City, UT 84112

ABSTRACT

The goal of the scene labeling problem is to identify a set of regions in a given image. There are several approaches to solve this problem, including backtracking, graph matching, etc. A new method called *split-level relaxation* based on discrete relaxation was proposed in [3]. It takes care of multiple semantic constraints, by considering each of them independently. The problem is NP-complete, so it takes a long time to solve this problem. With the advent of multiprocessors, it is now imperative to see if the problem can be solved faster in the average case.

We give a framework for solving the problem in a parallel processing environment, using split-level relaxation. Experiments done on a multiprocessor show that indeed it is advantageous to use them to solve this problem. The results are also presented.

1 Introduction

The scene labeling problem can be formulated as a consistent labeling problem. Here we are given a set of units, **U**, and a set, **D**, of possible labels for these units. A unit can be assigned any label from **D**. In general, however, there are restrictions on the labels a set of units can have simultaneously. These restrictions are represented as a relation, called the constraint relation, **R**. If $((u_i,l_i),(u_j,l_j)) \in \mathbf{R}$, then the unit u_i cannot have the label l_i, when u_j has the label l_j. The goal of the consistent labeling problem is to find a labeling which is consistent.

Since the problem is NP-complete[5], the algorithms that solve it are exponential in the worst case. Among the ways to solve this problem are generate-and-test, the standard backtracking method and several variations of it. Often a preprocessing step before backtracking is useful[1,2,4,5,6]. Mackworth[4] proposed node, arc and path consistency algorithms to remove labels which do not satisfy a minimum of consistency constraints. These algorithms do not produce a solution, but they prune the search space. This has been shown to be useful in vision applications[3].

Despite these efforts the overall time taken to label a scene is still very high. With the advent of multiprocessors it is necessary to examine the suitability of algorithms on these machines and also to explore new techniques that will be amenable to parallel processing. We present a new framework for scene labeling in a parallel processing environment.

[1]This work was supported in part by NSF Grants MCS-8221750, DCR-8506393, and DMC-8502115.

2 Scene Analysis as CLP

One approach to solve this problem is to first locate the features (e.g., holes, corners, etc.) in the image. Then these features and the constraints between them are used to identify the objects. Thus, the features in the scene constitute the set of units, **U**, the features of the possible objects in the scene are the labels, **D**, and the relations between them define the constraint relation, **R**. We restrict ourselves to binary constraints only. An important observation is that even though all the constraints are binary, they are not always the same. For example, consider two holes H_1 and H_2 in an image, such that H_1 is both adjacent to and bigger than H_2. The two constraints can be expressed as binary predicates *adjacent*(H_1, H_2), and *bigger*(H_1, H_2). Here even though both are binary constraints, they are of different *type*. The *adjacent* constraint is symmetric while the *bigger* is not. The semantics of the two constraints are also very different. Hence, we group the constraints according to their types.

It is easy to realize the consistent labeling problem using a graph model. The units are represented by the nodes of the graph and the arcs denote the constraints. However, what we have here is conceptually closer to a set of graphs rather than a single graph. Each graph models a single relation, which represents only one type of constraint. The topology of all these graphs remains unchanged during the relaxation process. What changes is the label sets of the nodes in the image graph. The model graphs are used to verify the constraints.

3 Split-Level Relaxation

The main reason to change the structure of the standard relaxation process is that there is no way to use positive information. For example, if at start we have the information that the primitive X can be either A or B, we can't use this information in any way more than if we did not have this information. In standard relaxation all the nodes are treated equally and the positive information which is very powerful, cannot be used. In fact, it is possible for X to lose all its labels during relaxation just because it lacks support at some other primitive which may not even belong to the model.

In split-level relaxation we divide the nodes (primitives) into two groups: nodes whose labels are determined using the positive information process are called *strong* nodes, and the rest are called *weak* nodes. The strong nodes signify positive information.

They always remain strong during the relaxation process, while the *weak* nodes may be elevated to the *strong* status. During relaxation a strong node can affect the label sets of other (both strong and weak) nodes. The weak nodes, however, may not change the label sets of strong nodes. This prevents the nodes which are not part of the model from affecting the label sets of strong nodes. Those nodes which survive the first few iterations, i.e., those whose label set is non-empty, are then made strong nodes. Those which do not survive are not used further and are considered not to be a part of the model. After that the relaxation process works as before. This new method is called the *Split-Level Relaxation.*

4 Parallel Split-Level Relaxation

There are several levels of parallelism in this relaxation procedure. At the top level it is possible to look for objects in different parts of the same image. It can be done systematically and efficiently as shown in [3]. The scene is divided into smaller subscenes and relaxation run in parallel in each of them.

In each of the subscenes we hypothesize that it contains one of several models and try to determine which object is actually present. These independent hypotheses can be verified concurrently. Each process checks to see if a particular object is present or absent in the subscene. The **split-level-relaxation** procedure in Figure 1 shows how this can be done. The $||for$ construct creates multiple tasks which execute concurrently.

In the next level we enforce consistency in the network of graphs. Essentially the graphs are independent, since they represent different types of constraints. So, the consistency tests can be performed in each of the graphs simultaneously. If there are four types of constraints, four processes work concurrently trying to enforce consistency. This is done by the **relax** procedure in Figure 1. It creates r processes, where r is the number of constraint types. The procedure *Consistency* enforces the consistency in the k^{th} graph. If any label is deleted while doing the consistency checks, the *change* flag is set to $TRUE$.

At the bottom level, consistency tests are made on a particular graph. It has been shown[7] that there is lot of parallelism in the node, arc, and path consistency algorithms that can also be exploited effectively on a parallel processor.

It is easy to show that this scheme works correctly. It can be done by proving the correctness of the **relax** procedure. **relax** always terminates and give the correct answer (which is always the same as one produced by a sequential algorithm). If a label l is removed from the label set of any node u during the **relax** process, then the node u cannot have the label l in any complete consistent labeling. Also, if a label is *not* removed at any node, then it doesn't violate any consistency tests.

Clearly there is a lot of parallelism in the split-level relaxation method. The number of processors required to exploit the total parallelism is also very large, even though it is still polynomial in terms of the node and domain size. The arc consistency procedures can effectively use up to n^2m^2 processors, where n is the number of nodes and m is the number of labels per node. So the **relax** procedure can use up to n^2m^2r processors, where there are r different graphs. The **split-level-relaxation** procedure has two more levels of parallelism, but the processor requirement depends on the actual image content. Although the number of processors required to get maximum parallelism is high, one can get very good speedup using fewer processors as well.

```
split-level-relaxation()
begin
    build-model-graphs();
    repeat
        ||for each subscene S_i ∈ S do
            O = Hypotheses();
            ||for each O_j ∈ O do
                relax(S_i,O_j);
    until (! done);
end

relax(Scene,Model)
begin
    build-image-graph(Scene);
    change := TRUE;
    while (change) do
        ||for graph-type k := 1 to r
            change := FALSE;
            Consistency(Scene,Model,k);
    end
end
```

Figure 1: Figure 1 : Parallel Split-Level Relaxation

5 Implementation and Results

The results in this paper were obtained on the BBN Butterfly parallel processor. The Butterfly is a homogeneous, shared-memory *MIMD* machine. Each processor node consists of an MC68020 microprocessor, a processor node controller, and up to 4 megabytes of memory. Our machine has 18 processors, with all but two nodes with 1MB of memory. All the processor nodes are connected through a network called the Butterfly switch, which is a self-routing, packet-switched Ω network.

We use a *queue* to keep all the tasks that need to be executed. Each process waits for a task in the queue. Once it finds a task, it dequeues it, and works on the specified task. The granularity of the task should be carefully chosen so as to maximize the efficiency of the system. Since there is so much parallelism in the algorithm, there are several ways to divide the tasks in our algorithms. The tasks in our implementation are neither very fine nor very coarse grained. A task in the system enforces consistency along one arc in a particular type of graph. So, if there are r constraint-types and e arcs, there are re tasks. Parallel AC-1 is the underlying arc consistency algorithm.

5.1 Test Problems

Three types of graphs are used to measure the performance of split-level relaxation. In the cyclic graph problem[7], the nodes of the graph are connected to form a cycle. The label set of node i is given by $\{ x : jn + i, 1 \leq j \leq m \}$, where there are m labels and n nodes. If the constraint used is the *greater-than* relation, only one label is removed from the network in one iteration and

hence it takes mn iterations to converge. In addition to *greater than* we use two more constraint types.

A complete graph is used to model the n-queens problem. The constraint is that no two queens should attack each other. If two queens in i^{th} and j^{th} row are in columns k and l, respectively, the constraints are: $k \neq l$ and $|i-j| \neq |k-l|$. In order to drive the relaxation process we fix the position of two queens (n^{th} and $n-1^{st}$) such that they conflict.

The number of arcs for the first two problems are $\mathbf{O}(n)$ and $\mathbf{O}(n^2)$, respectively. To be complete, we test some graphs whose number of arcs is $\mathbf{O}(n^{1.5})$. The arcs of the graphs are connected at random.

5.2 Results

The major yardstick used to measure the performance of the parallel algorithm is *speedup*. It is defined to be the ratio of the time taken by an algorithm on one processor to the time taken by it on k processors. Ideally, one would like to get a linear speedup with a slope of one. The speedup as a function of the number of processors for the three problems is given in Figures 2, 3, and 4. The results are shown for $n = 20$, 30, 40, and 50.

For the random graph, the maximum speedup obtained was about 13, using 16 nodes. In the other two problems we achieve *super-linear* speedup, i.e., the speedup is greater than number of processors. It happens in due to of combinatorial implosion. Since the processes are working concurrently, a change effected by any one of them is immediately seen by the others, since the memory is shared. This may save some work for the parallel algorithm. For smaller problems, the speedup is not very good because of fewer tasks and high overhead.

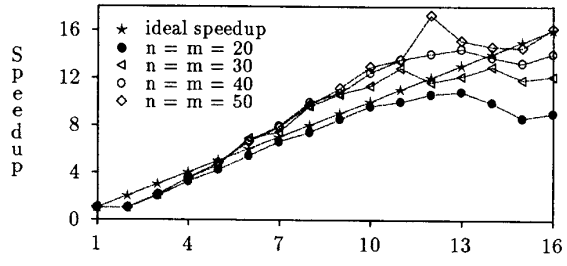

Figure 2: Speedup figures for the cyclic graph

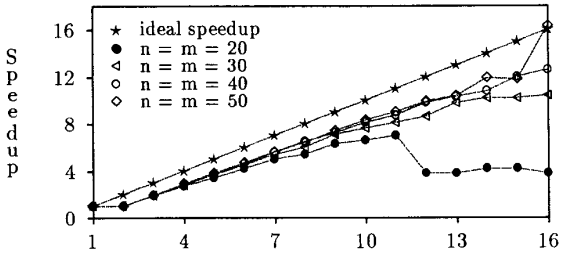

Figure 3: Speedup figures for the complete graph

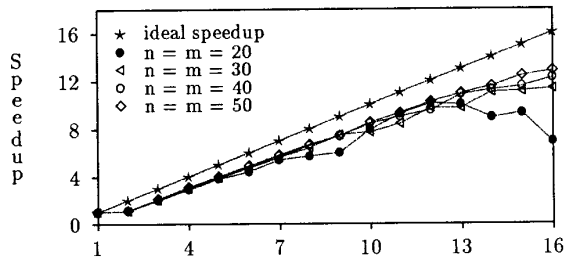

Figure 4: Speedup figures for the random graph

6 Conclusion and Future Research

In this paper we analyzed the split-level relaxation technique in a parallel processing framework. It was shown that there is much parallelism inherent in the algorithm that can be exploited. The algorithm was implemented on an actual multiprocessor and the results confirm this.

Although the results are good, the implementation can be made more efficient. Using multiple queues instead of a centralized one will reduce memory contention, particularly in large multiprocessors. An asynchronous implementation will also improve the performance. Another direction for future research is to explore parallelism in the path consistency algorithms.

References

[1] John Gaschnig. *Performance Measurements and Analysis of Certain Search Algorithms.* PhD thesis, Carnegie-Mellon University, Department of Computer Science, May 1979.

[2] Robert M. Haralick, Larry S. Davis, Azriel Rosenfeld, and David Milgram. Reduction Operations for Constraint Satisfaction. *Information Sciences*, 14:199–219, 1978.

[3] T.C. Henderson and Ashok Samal. Multi-constraint Shape Analysis. *Image and Vision Computing*, 4(2):84–96, May 1986.

[4] Alan K. Mackworth. Consistency in Network of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[5] Ugo Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences*, 7:95–132, 1974.

[6] Azriel Rosenfeld, Robert A. Hummel, and Steven W. Zucker. Scene Labelling by Relaxation Operations. *IEEE Transactions On Systems, Man, And Cybernetics*, SMC-6(6):420–433, June 1976.

[7] Ashok Samal and Thomas C. Henderson. Parallel Consistent Labelling Algorithms. *International Journal of Parallel Programming*, To appear, 1988.