Optimization techniques for nondeterministic promotion in the Andorra Kernel Language

(Extended Abstract)

Remco Moolenaar and Bart Demoen

Department of Computer Science
Celestijnenlaan 200 A, 3001 Heverlee, Belgium
{remco,bimbart}@cs.kuleuven.ac.be

In this extended abstract some optimization techniques are described for the nondeterministic promotion step in the Andorra Kernel Language [2]. First, a short introduction is given to the nondeterministic promotion step in AKL, second our parallel AKL implementation called ParAKL is described, third, the optimization techniques are described, and last but not least, a parallel algorithm is given for the nondeterministic promotion step.

The nondeterministic promotion step

The nondeterministic promotion step is an essential part of AKL: if a branch of the computation tree is stable (that is, the whole branch is suspended and no computation outside this branch could restart computations inside this branch), a predicate is selected for nondeterministic promotion. The guards of the alternatives of this predicate must be solved and the guard operator of this predicate must be the wait guard operator. The nondeterministic promotion step involves copying the surrounding and-box of the selected predicate to the and-boxes of the guards of this predicate. After this copying, the deterministic promotion rule is applicable for every alternative of this predicate.

The parallel AKL implementation

Our parallel AKL implementation called ParAKL [3] is based on the Warren Abstract Machine [6] and the PEPSys hashing scheme [7]. A branch consists of branchpoints and AND-branches. A branchpoint is allocated for every predicate with more than one alternative. An AND-branch is allocated for every clause with more than one goal. An offspring from a branch is separated from this branch using a hashwindow. In this hashwindow all the bindings made in the offspring to variables created outside the offspring are saved. All the other bindings can be done in place. The major difference between the original computation model and ParAKL is the subdivision of an and-box into a number of AND-branches. One of the optimization techniques is based on this.

Optimization techniques

The optimization techniques for the nondeterministic promotion step can be divided into two categories: predicate selection and copying.

Predicate selection techniques

Predicate selection is the first part of nondeterministic promotion. It selects the predicate on which the nondeterministic promotion rule will be applied. Therefore, selecting the right predicate could give rise to early failure, pruning of alternatives, etc. The following general techniques are applicable [4][5]:

1. Select the predicate with the least number of alternatives. This technique has one extra advantage: the amount of copying is limited.

2. First failure principle.

3. Select the predicate which is the most constrained predicate, that is, the predicate with the largest number of constraints on every argument.

Beside these general techniques, we developed some techniques particular to AKL:

4. Select the predicate with the most external binded variables. An external binded variable, is a variable that has a binding from one of the surrounding and-boxes of the predicate.

5. Select the oldest predicate. If two predicates are equally suited, choose the oldest one. This predicate probably was in a previous nondeterministic promotion step a possible candidate as well. So, instead of copying this predicate again, select it.

Our selection algorithm is, of course, a combination of these techniques. The first experimental results show that combining technique 3 and 4 can save at least 30 (compared to the leftmost selection rule of the AKL implementation from SICS). Other tests show that selecting the predicate with the most external binded variables (technique 4) can prune infinite branches from the computation tree.

Copying optimization techniques

Copying a part of the tree is an expensive operation. A lot of effort should therefore be put in speeding up this part of the nondeterministic promotion step. There are in general two ways to do this: speed up the copy algorithm itself or select a predicate which has to copy the smallest subtree. We have found optimizations for each one:

1. Copy only that part of the surrounding and-box that can be affected by the nondeterministic promotion. For ParAKL, this means: instead of taking the topmost AND-branch of the surrounding and-box, take the first AND-branch for which an argument of the selected predicate is subsumed by one of the local variables of the AND-branch. For instance:

```
:- a(X), b(Z).
a(X) :- c(X,Y), d(Y).
```

   If c/2 is the selected predicate, d/1 is copied, but not b/1. The test for subsumption of the arguments is only needed for arguments that do not have a binding that is external for the predicate. If all of the arguments have a binding that is ground and all these bindings are external for the predicate, nothing is copied.

2. If the arguments of a predicate are all ground and external, create a linked list containing the copied versions of this predicate and the original. If one of these predicates is selected for nondeterministic promotion, apply the outcome (fail or success) to all the predicates in the list. Such a predicate can be seen as a test predicate: the outcome of this predicate is the same for all the predicates in the list. This optimization is, in fact, a specialized version of the first one. Selecting a predicate is obsolete and some copying is saved.

As can be seen from the two optimizations, combining these optimizations with selection technique 4 can save a lot of copying. Experimental results, which are not available at the moment, could prove this.

A parallel algorithm for nondeterministic promotion

The sequential nondeterministic promotion step can be parallellized easily. It is not only easy, but it is necessary as well: it saves time and all the tasks that are created are moved to other processors, instead of kept on one processor. This improves load partitioning and, thus, overall efficiency. This algorithm uses the following concepts:

- Every processor has a list with all the possible candidates. A branchpoint is saved in this list, when, during guard evaluation, one of the guards is solved and its operator is the wait guard operator.
- If a branch needs to be copied during nondeterministic promotion, a task, containing the branch, is pushed on the goal stack. Other processors can take this task.

The outline of the parallel algorithm is as follows [1]:

```
procedure Nondeterministic_Promotion
begin
        cobegin
                Select local candidate from local candidate list;
        coend
        Select candidate from local candidates;
        Find topmost AND-branch;
        cobegin
                For every alternative of the candidate:
                        copy tree under topmost AND-branch;
        coend
end

procedure Copy_Tree
begin
        For every processblock of the branch
        begin
                Copy processblock;
                cobegin
                        For every branch of the processlist: Copy tree;
                coend
        end
end
```

Conclusions

We have presented some optimization techniques for the nondeterministic step in AKL. At this moment no complete experimental results can be given, but the first preliminary results show that these optimizations can lead to a more efficient implementation. As stated before, some preliminary results show that selection technique 4, which selects a predicate with the largest number of bound arguments, can prune possible infinite branches. As the copying algorithm uses this selection technique as a basis for its efficiency, it is without doubt, that this technique must be an essential part of the selection algorithm.

References

[1] E.W. Dijkstra, "Cooperating Sequential Processes", in Programming Languages, pp. 43-112, ed. F. Genuys, Academic Press, New York, 1968.

[2] Sverker Janson, Seif Haridi, "Programming Paradigms of the Andorra Kernel Language", in Proceedings of the International Logic Programming Symposium, pp. 167-183, ed. V. Saraswat, The MIT Press, Cambridge, Massachusetts, 1991.

[3] Remco Moolenaar and Bart Demoen, "A parallel implementation for AKL", submitted to PLILP '93, 1993.

[4] Jeffrey D. Ullman, "Principles of Database Systems", Computer Science Press, Rockville, 1982.

[5] Van Hentenryck, "Constraint Satisfaction in Logic Programming", The MIT Press, Cambridge, Massachusetts, 1989.

[6] Warren, David H.D., "An abstract Prolog instruction set", Technical Report no 309, p. 30, SRI International, Menlo Park, 1983.

[7] Harald Westphal, Philippe Robert, Jacques Chassin de Kergommeaux and Jean-Claude Syre, "The PEPSys Model: Combining Backtracking, AND- and OR-parallelism", in Proceedings of the 4th Symposium on Logic Programming, pp. 436-448, IEEE, 1987.