STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 2, 2002

A COST MODEL FOR THE AND-PARALLEL EXECUTION OF LOGIC PROGRAMS

MONICA VANCEA AND ALEXANDRU VANCEA

Abstract. Almost all the results regarding the automatic parallelization of logic programs assume ideal execution environments, focusing only on implicit parallelism detection and not taking into account practical computing system overheads. Trying to overcome such a drawback, we propose in this paper a cost model for the AND-parallel execution of logic programs, which is able to insert at compile time some cost functions which will estimate at run time the parallel execution costs involved. The cost functions are defined based on the particular computing system properties combined with the parallelization process features. If the conditions evaluated by these cost functions are met, the program is allowed to proceed in parallel. If not, it means that parallel execution may even require extra time compared with the sequential execution, so the code will be executed sequentially. We believe that our model is of a very practical importance allowing the run time environment to take the adequate decision with respect to the possibility of AND parallel execution of the (implicit) parallelism present in the logic programs.

1. Preliminaries

Automatic parallelization is the most suitable technique at the moment for exploiting the inherent parallelism from programs written in logic programming languages [Sehr92].

Among the parallel execution models employed at the level of logic programming languages, the AND parallel model raises the most complex problems [Chass84, Herm89, Lin88]. That is because this model is confronted with data dependence relations that appear frequently between the subgoals of a clause [Chang85, Shen92]. That is why, even if from a technically point of view we succeed to solve the actual automatic parallelization problem, the practical costs assumed by such an action (involving intensive data dependence testing followed by the generation of the equivalent parallel code) could be prohibitive.

2000 *Mathematics Subject Classification.* 68Q17, 68Q10, 68M20.

1998 *CR Categories and Descriptors.* D.1.6. [**Software**]: Programming Techniques – *Logic Programming*; D.1.3. [**Software**]: Programming Techniques – *Parallel Programming*; D.2.9. [**Software**]: Software Engineering – *Cost Estimation*; H.3.4. [**Information Systems**]: Information Storage and Retrieval – *Performance evaluation*.

When developing and analyzing parallel logic execution models we assume in general an ideal execution environment, making abstraction of a significant number of supplimentary practical costs (overheads). As examples of such overheads we mention here the task creation overhead, the time cost of task switching between processors, communication costs etc. The drawbacks implied by such practical execution factors may lead not only to the diminishing of the parallel execution performance, but even to a parallel execution time much greater than the corresponding sequential one! If such a situation is met, then parallel processing becomes an inappropriate decision for the program's execution.

In logic programming these issues are much more present in the case of AND parallelism where the possible data dependencies and their management may imply significant costs compared to other types of logic parallelism (like OR parallelism for example where such issues are not so critical [Chass94, Lusk90]).

These are the reason for which we propose in this paper a cost model for the AND parallelization of a sequential logic program. By applying this model we can estimate if the implied parallelization effort and the envisioned execution costs may keep also in practice the theoretical advantages of the parallel execution on the sequential one.

More exactly, we will propose ***a cost model for deriving sufficient conditions for deciding if the AND parallel execution of a particular logic program is an appropriate decision***.

Execution costs control can't be performed entirely as a compile time activity because in the general case these costs depend on the input data. On the other hand, a cost analysis performed entirely at run time risks to conclude non parallel execution in a great number of practical cases, taking into account a too large run time overhead. So, our strategy will be to divide in a reasonable way the workload between compile time and run time phases.

We will define and generate some *cost functions* at compile-time. Their mission will be to *estimate* at run time the total cost of parallel execution relatively to the particular size and nature of the input data, information which will be known at that moment, so possible of be practically taken into account.

2. Definitions and notations

Let S be the goal which we want to be analyzed and let suppose that it is composed of the (sub)goals $(s_1, \ldots, s_n)$ so $S = (s_1, \ldots, s_n)$. For the goal S we denote by:

- C_{seq} – the cost of its sequential execution;
- C_{par} – the cost of its parallel execution.

And we denote by C_i the execution cost for the s_i goal.

The sequential execution of the goal S may be performed only in one way, implying only obbeying the sequential execution order of goals $s_1, \ldots, s_n$ for the constituent subgoals. The parallel execution however, may be performed in many ways, its particular history and development depending on many influences, among these being the number of available processors, the implemented scheduling and memory allocation techniques etc. So when we refer to the cost of the sequential execution it is obvious what we mean, because this is a unique value at the level of a particular computing system. But we cannot refer to a single well defined value when we refer to the cost of the parallel execution, because it can follow diverse paths, one particular execution being selected upon some dynamic criteria.

For this reason and for our analysis to be enough general we will denote by *Cpar* the maximum cost of all possible parallel execution alternatives, that is, *the cost of the most costly parallel execution possibility for goal S.*

Our analysis intends to establish if the subgoals $s_1, \ldots, s_n$ justify their parallel execution. More exactly, from the viewpoint of the parallel execution opportunity, the cost analysis has to verify if the relation $C_{par} \leqslant C_{seq}$ holds or not.

Because of the way in which a parallel execution proceeds (as mentioned above based first of all on dynamic decisions) we cannot really *compute* the value C_{par}, but we have to *estimate* and/or *approximate* it (the notion of *execution time* itself assumes that the exact relation between C_{par} and C_{seq} can be established only after the execution of goal S).

We must decide further the approximation technique to be used. Let C_{par}^{sup} be an upper limit for the parallel execution cost (in fact we already assumed that $C_{par} = C_{par}^{\text{sup}}$) and let C_{seq}^{inf}be a lower limit for the sequential execution (the cost of the fastest possible sequential execution of the goal S).

Let's notice that if we succeed to prove for S that $C_{par}^{\text{sup}} \leqslant C_{seq}^{\text{inf}}$ then the same relation holds trivially between the actual execution times also, so running in parallel the subgoals $s_1,\ldots,s_n$ is a correct decision. Mathematically, the relation $C_{par}^{\text{sup}} \leqslant C_{seq}^{\text{inf}}$ becomes a sufficient condition for having $C_{par} \leqslant C_{seq}$, so $C_{par}^{\text{sup}} \leqslant C_{seq}^{\text{inf}}$ ***is a sufficient condition for the decision to run in (AND) parallel the subgoals of a given goal.***

3. Model description

We assume that we have k processors available for the execution of the n subgoals of the S goal.

Definition 3.1. We define ***the processor's average computing load*** as beeing the value $\Im_{md}^{=}\lceil n/k \rceil$, i.e. the number of subgoals that a processor must solve on the average.

Definition 3.2. We define as an upper bound of the total cost of the parallel execution

$$C_{par}^{\sup} = \Re_c^{\sup} + T_{par}^{\sup}$$

where $\Re_c^{\sup}$ is an upper bound for the creation cost of the tasks associated with the clause's subgoals (we will call it the *task creation overhead*) and $T_{par}^{\sup}$ is an upper bound for the parallel execution time taken by the goal S.

$\Re_c^{\sup}$is an architecture dependent value which can be experimentally determined. In general, such a value is a constant or a function depending on some parameters such as the number or size of the input data, the number of manageable tasks etc. We want in the following to approximate the value $T_{par}^{\sup}$.

Let $T_i^{\sup}$be an upper limit for the execution time of the goal s_i and let $T_{\max}^{\sup} = \max(T_1^{\sup}, \ldots T_n^{\sup})$. Obviously, we have then

$$T_{par}^{\sup} \leqslant m T_{\max}^{\sup}$$

Also, for every subgoal we have

$$T_i^{\sup} = P_i^{\sup} + C_i^{\sup}$$

where $P_i^{\sup}$represents the *scheduling overhead* for the goal s_i (the time passed between the corresponding task creation and the actual starting of its execution) and $C_i^{\sup}$ denotes *the effective run-time cost* for the goal s_i, without taking into consideration task creation overhead or the scheduling overhead.

Regarding the $T_{seq}^{\inf}$ value, this can be approximated as follows:

$$T_{seq}^{\inf} = T_{s_1}^{\inf} + \ldots + T_{s_n}^{\inf},$$

where $T_{seq(s_i)}^{\inf}$ denotes a lower bound for the cost of s_i's sequential execution (the best sequential execution for this subgoal).

All the above reasoning can be resumed by the following lemma.

Lemma 3.3. If the following relation holds

$$P^{\sup} + C_{par}^{\sup} \leqslant T_{s_1}^{\inf} + \ldots + T_{s_n}^{\inf}$$

then we have also $C_{par}^{\sup} \leqslant C_{seq}^{\inf}$.

This result can be further relaxed as we will show in the theorem 3.5.

Definition 3.4. By ***goal execution overhead*** we denote the total time taken by the corresponding task creation plus the time taken by the scheduling overhead for a particular goal, that is

$$\Re_e = \Re_c + \Re_P$$

where the goal scheduling overhead is approximated by $\Re_P = \Im_{md} \cdot P_i^{\sup}$.

The main result of this section is presented in theorem 3.5. and it establishes some sufficient conditions for the AND parallel execution of a logic program's clauses.

Theorem 3.5. Let $(s_1,\ldots,s_n)$ be the S goal's subgoals and let $m = \Im_{md}$. If among these subgoals there exists at least $m{+}1$ goals such as $\forall i = 1,\ldots, m{+}1$, $\Re_e \leqslant T_{s_i}^{\inf}$, then we have $C_{par} \leqslant C_{seq}$.

***Proof*.** If we have at least $m{+}1$ subgoals such that $\forall i = 1,\ldots, m{+}1$, $\Re_e \leqslant T_{s_i}^{\inf}$ then it follows that we have at least one subgoal s_j , $j = m{+}1,\ldots, n$, such that $\Re_e \leqslant T_{s_j}^{\inf}$, from where we conclude that even more it holds

$$\Re_e \leqslant T_{s_{m+1}}^{\inf} + \ldots + T_{s_n}^{\inf} \leqslant T_{s_{m+1}} + \ldots + T_{s_n}$$

By adding to the both members the running time for the goals $1\ldots m$ we obtain

$$\Re_e + T_{s_1} + \ldots + T_{s_m} \leqslant T_{s_1} + \ldots + T_{s_m} + T_{s_{m+1}} + \ldots + T_{s_n} \tag{1}$$

Let's recall that $m = \Im_{md}$ (the average computation load for a processor) indicates the average number of subgoals that will be sequentially processed by a processor during the parallel execution of the initial goal.

Because relation (1) holds for any m subgoals for which the execution time is present as a term in the left hand side, it holds in particular also for the case in which those m subgoals are those with the longest execution time. In this latter case, the left hand side of the inequality (1) obviously represents an upper bound for the parallel execution time of the initial goal S (because the parallel execution will take maximum the time taken by the sequential execution of the longest m subgoals at a processor plus the goal execution overhead for the entire S goal). It follows that we have

$$C_{par}^{\sup} \leqslant T_{s_1} + \ldots + T_{s_m} + T_{s_{m+1}} + \ldots + T_{s_n}$$

but the right hand side is nothing else than the sequential execution cost of the goal S, so we have

$$C_{par}^{\sup} \leqslant C_{seq}^{\inf}$$

which shows that the conditions which we gave as hypothesis are truly sufficient conditions for the AND parallel execution of logic programs.

Example 3.6. Let's consider the following program code sequence:

```
q([ ], [ ]).
q([H|T], [X|Y]) :-
        X is H + 1,
        q(T,Y).

r([ ], [ ]).
r([X|RX], [X2|RX1]) :-
        X1 is X * 2,
        X2 is X1 + 7,
        r(RX,RX1).
```

We will consider as an estimation of the execution cost (or more precisely as a unit of measure for this cost) of a goal the number of resolution steps required for proving it. Then the execution costs for the predicated q and r may be estimated upon the following cost functions:

```
Cost q(n) = 2n + 1
Cost r(n) = 3n + 1
```

We consider the AND parallel goal . . . q(X,Y) & r(X). . . expressed as in [Herm91] in which the argument list represents the set of the input arguments and not the arity of those predicates.

Based on the results of the theorem 3.5 the initial code sequence can be translated to

. . . length(X, LX), cost_q is LX2=1, cost_r is LX3+1,

(cost_q > Re(q), cost_r > Re(r) $\rightarrow$ q(X,Y) & r(X); q(X,Y), r(X)),. . .

where Re(q) and Re(r) denote respectively the parallel goal execution overhead and cost_q and cost_r represent the sequential execution costs for goals q and r respectively. The adnotation of the initial code sequence with such a condition allows in this moment to decide at run time whether or not to AND parallel execute the goals that follow the tested condition.

To conclude: the practical usefullness of our results from theorem 3.5 resides in the possibility to apply source code transformations at compile time which will insert the necessary tests to be performed at run time. These tests will decide upon the adequacy of running in AND parallel the adnotated sequence of goals.

4. Conclusions

Ideal execution environments are assumed when methods for automatic parallelization of logic programs are studied. In developing such methods, the focus is directed towards implicit parallelism detection and only very few models are taking into account the costs implied by practical computing system overheads. Estimating such costs are nevertheless of a critical importance because we can meet situations in practice for which parallel execution would be more time consuming than the equivalent sequential one. That is why we proposed in this paper a cost model for the AND-parallel execution of logic programs, which using adnotations capabilities inserts at compile time some cost functions which will perform at run time a good estimation of the parallel execution costs involved. The cost functions are defined and generated based on the computing system properties and on parallelization process features. If at run time the conditions evaluated by these cost functions are met, the program is allowed to proceed in parallel. If not, it means that parallel execution will not provide the expected speedup, so the code will be executed sequentially. We believe that our model is a very practical one, allowing the run time environment to take the adequate decision with respect to the possibility of AND parallel execution of the (implicit) parallelism present in the logic programs.

References

[Chang85] J-H. Chang, A.M. Despain, D. DeGroot, *And-Parallelism of Logic Programs based on Static Data Dependency Analysis*, in Digest of Papers of COMPCON, Spring 1985, pp. 218–225.

[Chass94] J. Chassin de Kergommeaux, P. Codognet, *Parallel Logic Programming Systems*, in ACM Computing Survey, vol.26, no.3, Sept.94, pp. 295–336.

[Herm89] M.V. Hermenegildo, F. Rossi, *On the Corectness and Efficiency of Independent AND-Parallelism in Logic Programs*, in Proc. of the 1989 North American Conf. on Logic Programming, 1989, pp. 369–389.

[Herm91] M.V. Hermenegildo and L.Greene, *The &-prolog System: Exploiting Independent And Parallelism*, New Generation Computing, 9 (3, 4), 1991 pp. 233–257.

[Lin88] Y. J. Lin, V. Kumar, *AND-parallel execution of Logic Programs on a shared Memory Multiprocessor: A Summary of Results*, in Fifth International Logic Programming Conference, Seatle, WA, 1988, pp. 1106–1120.

[Lusk90] E. Lusk, S. Haridi, D.H.D. Warren et. al., *The Aurora OR-Prolog System*, New Generation Computing, vol.7, no.2,3, 1990 pp. 243–273.

[Sehr92] D.C. Sehr, *Automatic Parallelization of Prolog Programs*, Ph.D. dissertation, Univ.of Illinois at Urbana-Champaign, 1992.

[Shen92] K. Shen, *Exploiting Dependent And-Parallelism in Prolog: The Dynamic, Dependent And-Parallel Scheme*, in Proc. Joint Int'l. Conf. and Symp. On Logic Prog. MIT Press, 1992, pp. 717–731.

Faculty of Economic Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: `vancea@econ.ubbcluj.ro`

Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: `vancea@cs.ubbcluj.ro`