Towards a General Framework for Static Cost Analysis of Parallel Logic Programs[*]

M. Klemen
IMDEA Software Institute
ETSIINF, U. Politécnica de Madrid
`maximiliano.klemen@imdea.org`

P. Lopez-Garcia
IMDEA Software Institute
Consejo Sup. Inv. Cient. (CSIC)
`pedro.lopez@imdea.org`

J.P. Gallagher
IMDEA Software Institute
Roskilde University
`john.gallagher@imdea.org`

J.F. Morales
IMDEA Software Institute
`josef.morales@imdea.org`

M.V. Hermenegildo
IMDEA Software Institute
ETSIINF, U. Politécnica de Madrid
`manuel.hermenegildo@imdea.org`

Estimating in advance the resource usage of computations is useful for a number of applications. Examples include granularity control in parallel/distributed systems, automatic program optimization, verification of resource-related specifications, and detection of performance bugs, as well as helping developers make resource-related design decisions. Besides *time* and *energy*, we assume a broad concept of resources as numerical properties of the execution of a program, including the number of *execution steps*, the number of *calls* to a procedure, the number of *network accesses*, the number of *transactions* in a database, and other user-definable resources. The goal of automatic static analysis is to estimate such properties (prior to running the program with concrete data) as a function of input data sizes and possibly other (environmental) parameters.

Due to the heat generation barrier in traditional sequential architectures, parallel computing, with (possibly heterogeneous) multi-core processors in particular, has become the dominant paradigm in current computer architecture. Predicting resource usage on such platforms poses important challenges. Most work on static resource analysis has focused on sequential programs, and comparatively less progress has been made on the analysis of parallel programs, or on parallel logic programs in particular. The significant body of work on static analysis of sequential logic programs has already been applied to the analysis of other programming paradigms, including imperative programs. This is achieved via a transformation into *Horn clauses*.

In this paper we concentrate on the analysis of parallel Horn clause programs, which could be the result of such a translation from a parallel imperative program or be themselves the source program. Our starting point is the standard general framework of CiaoPP for setting up parametric relations representing the resource usage (and size relations) of programs. This is based on the well-developed technique of setting up recurrence relations representing resource usage functions parameterized by input data sizes, which are then solved to obtain (exact or safely approximated) closed forms of such functions (i.e., functions that provide upper or lower bounds on resource usage). The framework is doubly parametric: first, the costs inferred are functions of input data sizes, and second, the framework itself is parametric with respect to the type of approximation made (upper or lower bounds), and to the resource analyzed. We build on this and propose a novel, general, and flexible framework for setting up cost equations/relations

[*]This document is an extended abstract of Technical Report CLIP-1/2019.0 [1]. Research partially funded by MINECO project TIN2015-67522-C3-1-R *TRACES* and Comunidad de Madrid project S2018/TCS-4339 *BLOQUES-CM*, co-funded by EIE Funds of the European Union.

which can be instantiated for performing static resource usage analyses of parallel logic programs for a wide range of resources, platforms, and execution models. Such analyses estimate both lower and upper bounds on the resource usage of a parallel program as functions on input data sizes. We have instantiated the framework for dealing with Independent And-Parallelism (IAP), which refers to the parallel execution of conjuncts in a goal. However, the results can be applied to other languages and types of parallelism, by performing suitable transformations into Horn clauses.

Independent And-Parallelism arises between two goals (or other parts of executions) when their corresponding executions do not affect each other. For pure goals (i.e., without side effects) a sufficient condition for the correctness of IAP is the absence of variable sharing at run time among such goals. (Restricted) IAP has traditionally been expressed using the `&/2` meta-predicate as the constructor to represent the parallel execution of goals. In this way, the conjunction of goals (i.e., literals) `p & q` in the body of a clause will trigger the execution of goals `p` and `q` in parallel, finishing when both executions finish.

Automatically finding closed-form upper and lower bounds for recurrence relations is an uncomputable problem. For some special classes of recurrences, exact solutions are known, for example for linear recurrences with one variable. For some other classes, it is possible to apply transformations to fit a class of recurrences with known solutions, even if this transformation obtains an appropriate approximation rather than an equivalent expression.

Particularly for the case of analyzing independent and-parallel logic programs, nonlinear recurrences involving the *max* operator are quite common. For example, if we are analyzing elapsed time of a parallel logic program, a proper parallel aggregation operator is the maximum between the times elapsed for each literal running in parallel. To the best of our knowledge, no general solution exists for recurrences of this particular type. However, in this paper we identify some common cases of this type of recurrences, for which we obtain closed forms that are proven to be correct.

We have implemented a prototype of our approach, leveraging the existing resource usage analysis framework of CiaoPP. The implementation basically consists of the parameterization of the operators used for sequential and parallel cost aggregation, i.e., for the aggregation of the costs corresponding to the arguments of `,/2` and `&/2`, respectively. This allows the user to define resources in a general way, taking into account the underlying execution model. We introduce a new general parameter that indicates the execution model the analysis has to consider. For our current prototype, we have defined two different execution models: standard *sequential* execution, represented by *seq*, and an abstract parallel execution model, represented by $par(n)$, where $n \in \mathbb{N} \cup \{\infty\}$. The abstract execution model $par(\infty)$ is similar to the *work* and *depth* model presented and used extensively in previous work. Basically, this model is based on considering an unbounded number of available processors to infer bounds on the depth of the computation tree. The *work* measure is the amount of work to be performed considering a sequential execution. These two measures together give an idea on the impact of the parallelization of a particular program. The abstract execution model $par(n)$, where $n \in \mathbb{N}$, assumes a finite number n of processors.

For the evaluation of our approach, we have analyzed a set of benchmarks that exhibit different common parallel patterns, together with the definition of a set of resources that help understand the overall behavior of the parallelization.The results show that most of the benchmarks have different asymptotic behavior in the sequential and parallel execution models. As mentioned before, this is an upper bound for an ideal case, assuming an unbounded number of processors. Nevertheless, such upper-bound information is useful for understanding how the cost behavior evolves in architectures with different levels of parallelism. In addition, this *dual* cost measure can be combined together with a bound on the number of processors in order to obtain a general asymptotic upper bound.

References

[1] M. Klemen, P. Lopez-Garcia, J. Gallagher, J.F. Morales & M. V. Hermenegildo (2019): *Towards a General Framework for Static Cost Analysis of Parallel Logic Programs*. Technical Report CLIP-1/2019.0, The CLIP Lab, IMDEA Software Institute and T.U. Madrid. Available at `http://arxiv.org/abs/1907.13272`.