Using Combined Static Analysis and Profiling for Logic Program Execution Time Estimation

Edison Mera[1], Pedro López-García[1], Germán Puebla[1],
Manuel Carro[1], and Manuel Hermenegildo[1,2]

[1] Technical University of Madrid
edison@clip.dia.fi.upm.es, {pedro.lopez,german,mcarro,herme}@fi.upm.es
[2] University of New Mexico, herme@unm.edu

Motivation. Predicting statically the running time of programs has many applications ranging from task scheduling in parallel execution to proving the ability of a program to meet strict time constraints. A starting point in order to attack this problem is to infer the computational complexity of such programs (or fragments thereof). This is one of the reasons why the development of static analysis techniques for inferring cost-related properties of programs (usually upper and/or lower bounds of actual costs) has received considerable attention.

In most cases such cost properties are expressed using platform-independent metrics: e.g., the number of resolution steps that a procedure will execute as a function of the size of its input data [2, 3]. Although platform-independent costs have been shown to be useful in various applications [6, 4], in distributed execution and mobile/pervasive computation scenarios involving hosts with different computational power, it becomes necessary to express costs in a way that can be instantiated later to different architectures, to accurately reflect execution time.

Approach. With this objective in mind, we have developed a framework which combines cost analysis with profiling techniques in order to infer functions which yield bounds on platform-dependent *execution times* of procedures [7]. In this framework, platform-independent cost functions, parameterized by a certain number of constants, are inferred for each procedure in a given program. These parameters aim at capturing the execution time of certain low-level operations on each platform which is assumed to be independent from data size. Their selection is, obviously, critical. For each execution platform, the value of such constants is determined experimentally by running a set of synthetic benchmarks and measuring their execution time with a profiling toolkit developed in-house. Once such constants are determined, they are substituted into the parametric cost functions to make it possible to predict, with a certain accuracy, actual execution times.

Each selection of parameters for the cost functions determines a cost model. We have implemented this approach in the CiaoPP system [5], and studied a number of cost models in order to determine experimentally which one is more precise. In doing this we have taken into account the trade-off between simplicity of the cost model (which affects the efficiency of the cost analysis and the complexity of the profiling) and the precision of their results. The results achieved show that the combined framework predicts the execution times of programs with a reasonable degree of accuracy and paves the way for more accurate analyses by including additional parameters. We believe this is an encouraging result, since using a one-time profiling for estimating execution times of other, unrelated programs is clearly appealing.

Further Applications. Deducing the expected execution time of programs in a fully automatic way has applications besides the already mentioned, more classical ones. For example, in a Proof-Carrying Code (PCC) framework, producers can send a certificate which includes a platform-independent cost function. The consumer can then, using a calibrating program, compute the values for the constants appearing in the parametric cost functions to obtain certified platform-dependent cost functions. Another application is found in resource-oriented specialization, where refined cost models can be used to help in guiding specialization by taking into account not only the size of the resulting program, but also its expected execution time (and maybe other low-level implementation factors). In particular, they can be used to perform self-tuning specialization in order to compare different specialized versions according to their costs [1].

References

1. S.J. Craig and M. Leuschel. Self-tuning resource aware specialisation for Prolog. In *Proc. of PPDP'05*, pages 23–34. ACM Press, 2005.
2. S.K. Debray and N.W. Lin. Cost analysis of logic programs. *ACM Transactions on Programming Languages and Systems*, 15(5):826–875, November 1993.
3. S.K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin. Lower Bound Cost Estimation for Logic Programs. In *1997 International Logic Programming Symposium*, pages 291–305. MIT Press, Cambridge, MA, October 1997.
4. M. Hermenegildo, E. Albert, P. López-García, and G. Puebla. Abstraction Carrying Code and Resource-Awareness. In *Proc. of PPDP'05*. ACM Press, July 2005.
5. Manuel V. Hermenegildo, Germán Puebla, Francisco Bueno, and Pedro López-García. Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). *Science of Computer Programming*, 58(1–2):115–140, October 2005.
6. P. López-García, M. Hermenegildo, and S.K. Debray. A Methodology for Granularity Based Control of Parallelism in Logic Programs. *J. of Symbolic Computation, Special Issue on Parallel Symbolic Computation*, 22:715–734, 1996.
7. E. Mera, P. López-García, G. Puebla, M. Carro, and M. Hermenegildo. Combining Static Analysis and Profiling for Estimating Execution Times in Logic Programs. Technical Report CLIP5/2006.0, Technical University of Madrid (UPM), School of Computer Science, UPM, April 2006.

Using Combined Static Analysis and Profiling for Logic Program Execution Time Estimation

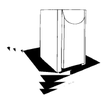
Edison Mera[1], Pedro López-García[1], Germán Puebla[1], Manuel Carro[1], Manuel Hermenegildo[1,2]

[1]School of Computer Science, Technical University of Madrid (UPM), Madrid, Spain.
[2]Depts. of Computer Science and Electrical Engineering and Computer Eng., University of New Mexico, Albuquerque, NM, USA.

Intuition

Platform independent compile-time cost analysis

Infers cost functions **parameterized by some constants**:
cost(p(X)) = K*length(X)



Platform dependent compile-time cost analysis

Functions that yield **execution times** depending on size of input:
cost(p(X)) = 0.55*length(X)

Prediction of execution times for concrete inputs

cost(p([1,2,3])) = 1.65

Platform dependent one-time profiling to calibrate constants

- calibrator_1(_)
- calibrator_2(_)
- calibrator_3(_)
- ...

K = 0.55

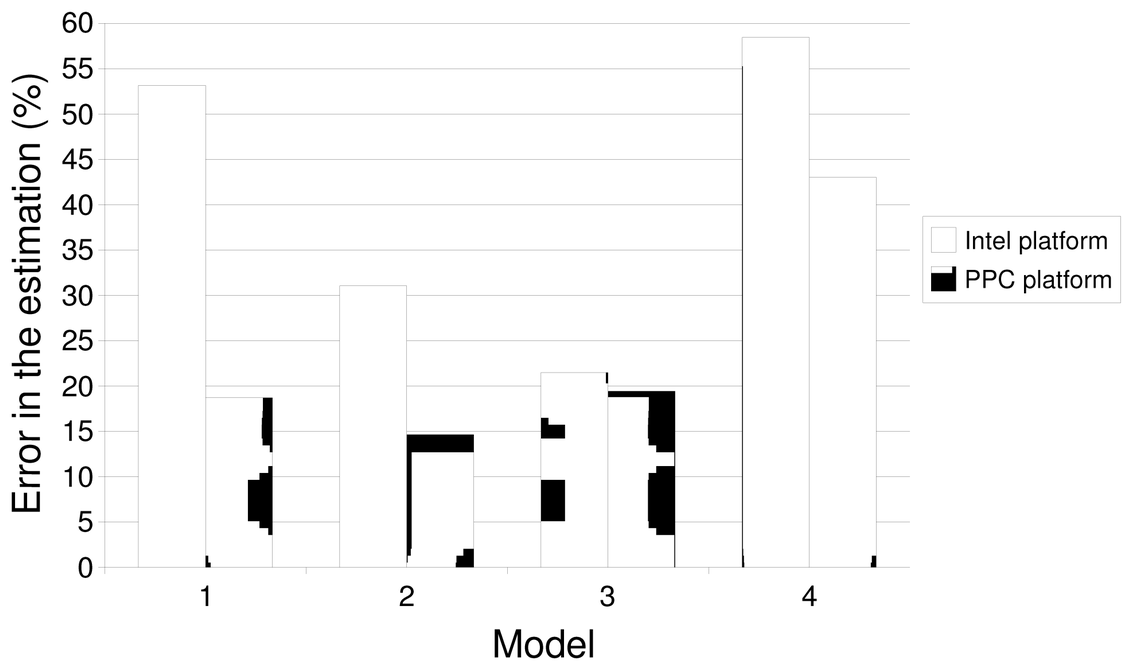
Assessment

Compare the estimated execution time with the observed time.

obs_time(p([1,2,3)) = 1,70
(3% error)

Assessment results

Cost model accuracy - Global comparison



Model (Ω)	Components
1	step nargs giunif gounif viunif vounif
2	step giunif gounif viunif vounif
3	step giunif gounif vounif
4	step

Applications

- Debugging/verification of timing properties. Guaranteeing meeting performance specifications.
- Resource/granularity control in parallel/distributed computing.
- Resource-oriented specialization. The inferred cost functions can be used to guide the specialization process.
- Mobile code safety and in particular Proof Carrying Code scenarios: e.g., code with timing guarantees in the ACC model.
- All currently implemented in CiaoPP!

Detailed Example

Platform-independent cost components define a cost model Ω.

Examples of cost components:

l(steps)=1 per clause.
l(nargs)=The arity of the clause head.
l(giunif)=Function symbols, variables and constants in the clause head which appear in input arguments.
l(gounif)=Function symbols, constants and variables in the clause head which appear in output arguments.
l(viunif)=Variables in the clause head corresponding to input arguments.
l(vounif)=Variables in the clause head corresponding to output arguments.

Program

```
:- module(nrev,[nrev/2],
      [assertions]).

:- entry nrev/2:list(int)*var.

nrev([],[]).
nrev([B|A],C) :-
      nrev(A,D),
      append(D,[B],C).

append([],A,A).
append([B|A],D,[B|C]) :-
      append(A,D,C).
```

Static Analysis

$\mathbf{Cost_p(l(\Omega),n)}=(Cost_p(l(\omega_1),\mathbf{n}),...,Cost_p(l(\omega_v),\mathbf{n}))$ where $\mathbf{l(\Omega)}=(l(\omega_1),...,l(\omega_v))$ and **n** is the size of input arguments.
The components ω_i define a particular cost model **Ω**

Result of platform dependent profiling

vector of constants, from running calibrators for **l(Ω)**

$\mathbf{K_\Omega}=(K_{\omega_1},...., K_{\omega_v})$
= (21.27,9.96,10.3,8.23,6.46,5.69)

Concrete input

p=nrev(A, B)
A=[1,2,3,4,5]
n=length(A)=5

Estimation of the execution time

$Exec\_time_p(\mathbf{n})= \mathbf{K_\Omega} \bullet Cost_p(\mathbf{l(\Omega)},\mathbf{n})$

ω_i	$Cost_p(\mathbf{l}(\omega_i),\mathbf{n})$	$Cost_p(l(\omega_i),5)$	K_{ω_i} (us)	K_{ω_i} x $Cost_p(l(\omega_i),\mathbf{n})$
steps	$0.5*n^2+1.5*n+1$	21	21.27	446.7
nargs	$1.5*n^2+3.5*n+2$	57	9.96	567.7
giunif	$0.5*n^2+3.5*n+1$	31	10.30	319.3
gounif	$0.5*n^2-0.5*n+1$	16	8.23	131.7
viunif	$1.5*n^2+1.5*n$	45	6.46	290.7
vounif	n^2+n	30	5.69	170.7
Execution time (ms)		**$K_\Omega \bullet Cost_p(l(\Omega),n)$**		**1926.8**

Conclusions

- Developed framework which allows estimating execution times.
- Combines static analysis with one-time profiling.
- Implemented and integrated in the CiaoPP system.
- The combined framework predicts the execution times of programs with a reasonable degree of accuracy.
- We believe this is an encouraging result, since using a one-time profiling for estimating execution times of other, unrelated programs is clearly a challenging goal.
- Interesting trade-off between accuracy and simplicity of the approach.
- Precision can be improved by using more refined cost models which take into account additional (lower level) factors.