Fully Automatic, Parametric Worst-Case Execution Time Analysis

Björn Lisper
Dept. of Computer Science and Engineering, Mälardalen University
P.O. Box 883, SE-721 23 Västerås, SWEDEN
bjorn.lisper@mdh.se

Abstract

Worst-Case Execution Time (WCET) analysis means to compute a safe upper bound to the execution time of a piece of code. Parametric *WCET analysis yields symbolic upper bounds: expressions that may contain parameters. These parameters may represent, for instance, values of input parameters to the program, or maximal iteration counts for loops. We describe a technique for fully automatic parametric WCET analysis, which is based on known mathematical methods: an abstract interpretation to calculate parametric constraints on program flow, a symbolic method to count integer points in polyhedra, and a symbolic ILP technique to solve the subsequent IPET calculation of WCET bound. The technique is capable of handling unstructured code, and it can find upper bounds to loop iteration counts automatically.*

1 Introduction

Parametric (or symbolic) WCET analysis derives a formula for the execution time, expressed in parameters of the program, rather than just a single number. The parameters can be either external, or internal like a symbolic upper bound to a loop count. A parametric WCET formula contains much more information than just a single WCET estimate, and it can be used for applications like online scheduling of tasks where parameters are unknown until runtime, or to find which parts of a code that has the strongest influence on the WCET. Thus, it is also potentially much more useful.

Previous approaches to parametric WCET have been based on the program timing schema model [4], or paths [1, 2]. These methods need manual annotations for constraints on loop counters and infeasible paths. An iterative method to compute parametric WCET bounds for simple loops has also been suggested [10].

Our method is potentially much more powerful than previous approaches. It can find loop bounds and infeasible path constraints automatically, and is capable of using such complex constraints in the calculation phase. Here we give a short account for the method followed by an explanatory example. A more detailed description is found in [8].

2 The Method

The analysis consists of a symbolic flow analysis, a symbolic summation, and a symbolic IPET calculation. See Fig. 1. The analysis uses a control flow graph model for programs, which means it works also for unstructured codes with jumps.

In the flow analysis, upper bounds for execution counts are derived in the form of symbolic expressions. If the program terminates, then the actual execution count for a program point equals the number of different states encountered in that point. Our method derives an upper approximation to the set of possible states in each program point, and then calculates the number of points in this set approximation. This yields an upper bound to the actual number of states and thus, under the assumption that the program terminates, the execution count. The set approximation may be parameterized: the counting is then performed symbolically. Program variables that affect the program flow, but do not change during the execution, are classified as parameters since varying their values will give rise to more states than are actually traversed during a single execution.

Set approximations can also be used to limit the number of states for which certain program paths can be taken. This can be used to find infeasible paths. It is also useful when analyzing program flows for pipelined processors, where different execution paths must be explored for possible pipeline overlap effects.

Sets of states in program points can be approximated from above by classical abstract interpretation [5]. Abstract interpretation is a framework that covers many possible ways to approximate sets of states. One abstract interpretation of particular interest is Halbwach's *polyhedral abstract interpretation* [6], which computes polyhedra as set approximations. Each program variable that may affect the program flow corresponds to a dimension in the polyhe-

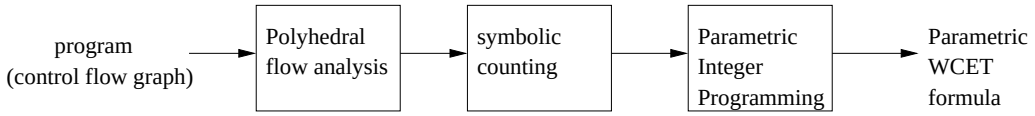

Figure 1. Structure of the method.

dral space. For instance, the set of states in a nested loop with loop indices i, j and upper (parametric) loop limits m, n will be bounded by a four-dimensional polyhedron in (i, j, m, n)-space.

Polyhedra are convex approximations. They describe linear loop index dependencies in nested loops, such as triangular loops, well but will overapproximate index sets for loops with non-unit strides. Other parametric set approximations are certainly possible, and will then provide different tradeoffs between precision and speed. Investigating these tradeoffs is an interesting topic of future research.

The next step is to count the numbers of points in polyhedra. Two techniques are known: through successive projection using known formulae for sums of powers of integers [9], and using *Ehrhart Polynomials* [3]. Both methods can compute parametric results. In our loop example above we would count points with respect to i and j, and return a sum that is parametric in m and n.

The final phase is the IPET calculation. It is done by *Parametric Integer Programming* (PIP) [7], which is a parametric extension of integer linear programming. This algorithm finds the optimum of a linear objective function over the parameterized set

$$\{\, \vec{x} \mid \vec{x} \geq \vec{0}, A\vec{x} + B\vec{z} + \vec{c} \geq \vec{0}, \vec{x} \text{ integral} \,\}$$

where $\vec{z}$ is a vector of parameters.

The parametric sums derived by the flow analysis are typically nonlinear in the parameters. However, each such sum can be replaced by a new symbolic parameter in the symbolic IPET calculation. The constraints will then become linear in these new parameters: PIP can compute an optimum expressed in them, and a subsequent substitution with the original sums followed by a simplification will yield the optimum expressed in the original parameters of the program. However, the new parameters often have direct interpretations, such as upper bounds to execution counts in program points, and can thus be interesting in their own right. An option is to leave them in the final answer. The resulting formula will then provide information how sensitive the WCET is for changes in loop counts and similar, which is interesting when tuning the code for best WCET.

The procedure outlined above is a fully automatic method for parametric WCET analysis that goes all the way from flow analysis to final WCET calculation. As far as we know, no other parametric method achieves this. The parametric calculation generalizes conventional IPET and can deal with advanced architectural features such as pipelining and caches in the same way.

3 An Example

Consider the CFG in Fig. 2. We assume each node n_i in the CFG has an execution time t_i. Each arc i has an execution count x_i. We first analyze it in order to extract upper bounds for the execution counts for all statements. It suffices to analyze the program w.r.t. the possible values of i and n, since they are the only variables affecting the program flow. We assume that B1 and B2 are basic block that update neither i nor n. There is one polyhedron S^i for each program point i, however $S^6 = S^4$ and $S^7 = S^5$ since B1 and B2 touch neither i nor n. We must also have $S^8 = S^3$. The system is solved by *fixed-point iteration* for the simplified system, which converges in nine iterations. The result is shown in Table 1. ($\top$ stands for the universal set: $S^0 = \top$ thus means that we allow any starting state in the analysis.)

We now calculate the number of points in the polyhedra. In the CFG in Fig. 2, the conditions depend on i and n only. n is never updated, and is thus considered a parameter. For each node, the number of elements in the abstract state on the preceding edge provides an upper bound on the execution count. The execution count for node n_0 is trivially one. By the method in [9], we obtain:

$$\begin{aligned}
\left|S^2\right| &= \infty \\
\left|S^3\right| &= \left|S^8\right| = \mathit{if}\ \ n > 0\ \ \mathit{then}\ \ n\ \ \mathit{else}\ \ 0 \\
\left|S^4\right| &= \mathit{if}\ \ n \geq 11\ \ \mathit{then}\ \ n - 11\ \ \mathit{else}\ \ 0 \\
\left|S^5\right| &= \mathit{if}\ \ n \geq 1\ \ \mathit{then} \\
&\qquad \mathit{if}\ \ n \geq 10\ \ \mathit{then}\ \ 10\ \ \mathit{else}\ \ n \\
&\quad \mathit{else}\ \ 0
\end{aligned}$$

This yields bounds $x_i \leq \left|S^i\right|$, where x_i is execution count for basic block n_i.

S^2 is overapproximated in the analysis. Therefore, there is no upper bound for x_2. This may seem awkward. However, there are *structural flow constraints* on the execution counts in addition to the upper bounds: for any node in the CFG, the sum of the execution counts for the input arcs must

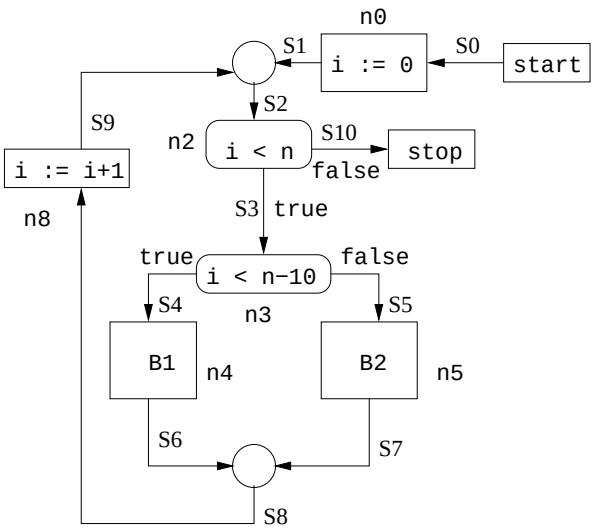

Figure 2. A simple flowchart program.

S^0	S^1	S^2	S^3	S^4	S^5	S^9	S^{10}
$\top$	$i = 0$	$i \geq 0$	$i \geq 0$ $i \leq n-1$	$i \geq 0$ $i \leq n-11$	$i \geq 0$ $n-10 \leq i \leq n-1$	$i \geq 1$ $i \leq n$	$i \geq 0$ $i \geq n$

Table 1. Result of abstract interpretation for example flowchart.

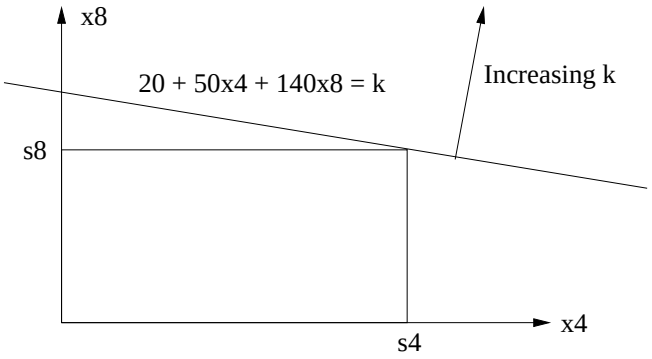

Figure 3. Maximizing the objective function in the IPET example.

equal the corresponding sum for the output arcs. These constraints will ensure finiteness of x_2, see below.

We finally perform a parametric IPET calculation. The WCET estimate is $\max(\sum_{i=0,2,3,4,5,8} x_i t_i)$. The execution count bounds derived from the abstract interpretation yield constraints $x_i \leq s_i, \quad i = 1, \ldots, 10$, where s_i is a symbolic parameter for $\left|S^i\right|$. We also have non-negativity constraints $x_i \geq 0, \quad i = 1, \ldots, 10$.

The structural flow constraints of the CFG can be used to reduce the number of variables down to two. Selecting x_4 and x_8 as basis yields the reduced problem $\max(t_0 + t_2 + x_4(t_4 - t_5) + x_8(t_2 + t_3 + t_5 + t_8))$ under the constraints $0 \leq x_4 \leq s_4$, $0 \leq x_8 \leq s_8$. Let us assume computation times $t_0 = 10$, $t_2 = 10$, $t_3 = 20$, $t_4 = 150$, $t_5 = 100$, and $t_8 = 10$. We then obtain the WCET estimate $50s_4 + 140s_8 + 20$ for $x_4 = s_4$, $x_8 = s_8$. See Fig. 3. With s_4, s_8 as the functions of n given by the polyhedral abstract interpretation we obtain (after simplification):

$$\begin{aligned} n \geq 11: &\quad 190n - 530 \\ 0 < n \leq 10: &\quad 140n + 20 \\ \text{otherwise}: &\quad 20 \end{aligned}$$

4 Conclusions and Further Research

We have described and exemplified a fully automatic method for parametric WCET calculation, that can deal with complex flow constraints and advanced architectural processor features. Future work involves a full implementation in order to evaluate the method with respect to accuracy and practical time complexity.

References

[1] G. Bernat and A. Burns. An approach to symbolic worst-case execution time analysis. In *Proc. 25th Workshop on Real-Time Programming*, Palma, Spain, May 2000.

[2] R. Chapman. Worst-case timing analysis via finding longest paths in SPARK Ada basic-path graphs. Technical Report YCS246, The British Aerospace Dependable Computing System Centre, Dept. of Computer Science, Univ. York, Oct. 1994.

[3] P. Clauss. Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyze and transform scientific programs. In *Proc. International Conference on Supercomputing*, pages 278–285, Philadelphia, PA, 1996. ACM.

[4] A. Colin and G. Bernat. Scope-tree: a program representation for symbolic worst-case execution time analysis. In *Proc. 14th Euromicro Conference on Real-Time Systems*, Vienna, June 2002.

[5] P. Cousot and R. Cousot. Abstract interpretation: A unified model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[6] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM Symposium on Principles of Programming Languages*, pages 84–97, 1978.

[7] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22:243–268, Sept. 1988.

[8] B. Lisper. Fully automatic, parametric worst-case execution time analysis. MRTC report, Dept. of Computer Science and Engineering, Mälardalen University, Apr. 2003. `http://www.mrtc.mdh.se/publ.php3?id=0531`.

[9] W. Pugh. Counting solutions to Presburger Formulas: How and why. In *Proc. ACM SIGPLAN'94 Conference on Programming Language Design and Implementation*, pages 121–134, Orlando, FL, June 1994. ACM.

[10] E. Vivancos, C. Healy, F. Mueller, and D. Whalley. Parametric Timing Analysis. In J. Fenwick and C. Norris, editors, *Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'2001)*, pages 88–93, Snowbird, Utah, June 2001.