Generalizing Parametric Timing Analysis

Joel Coffman[1], Christopher Healy[1], Frank Mueller[2], and David Whalley[3]

[1] Computer Science Dept.
Furman University
Greenville, SC 29613
chris.healy@furman.edu

[2] Computer Science Dept.
North Carolina State University
Raleigh, NC 27695-7534
mueller@cs.ncsu.edu

[3] Computer Science Dept.
Florida State University
Tallahassee, FL 32306-4530
whalley@cs.fsu.edu

ABSTRACT

In the design of real-time and embedded systems, it is important to establish a bound on the worst-case execution time (WCET) of programs to assure via schedulability analysis that deadlines are not missed. Static WCET analysis is performed by a timing analysis tool. This paper describes novel improvements to such a tool, allowing parametric timing analysis to be performed. Parametric timing analyzers receive an upper bound on the number of loop iterations in terms of an expression which is used to create a parametric formula. This parametric formula is later evaluated to determine the WCET based on input values only known at runtime. Effecting a transformation from a numeric to a parametric timing analyzer requires two innovations: 1) a summation solver capable of summation non-constant expressions and 2) a polynomial data structure which can replace integers as the basis for all calculations. Both additions permit other methods of analysis (e.g. caching, pipeline, constraint) to occur simultaneously. Combining these techniques allows our tool to statically bound the WCET for a larger class of benchmarks.

Categories and Subject Descriptors

C.3 [SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS]: REAL-TIME AND EMBEDDED SYSTEMS

General Terms

Verification, Reliability

Keywords

Worst-case execution time (WCET) analysis, parametric timing analysis

1. INTRODUCTION

Embedded systems, especially those in safety-critical or hard real-time environments, typically require that timing constraints be met. To guarantee these systems will meet deadlines, the worst-case execution time (WCET) of each program must be known. The process of determining the WCET of a program is known as *timing analysis*. Knowledge of the WCET can be used to dynamically scale voltage when a scheduler detects future slack time [1, 6, 7]. This facilitates power savings, an especially important aspect in embedded systems. Image processing provides another useful application for parametric timing analysis, since image dimensions may not be known *a priori* [10].

Static timing analysis has traditionally required loops to contain a constant number of iterations so analyzers may produce constant worst case execution bounds. Such constraints on input programs make this form of timing analysis numerical: the number of loop iterations is constant as is the final result from the timing analyzer. Parametric timing analysis allows the number of loop iterations to be unknown at compilation as long as this value may be written as an expression. Such flexibility expands the class of programs which may be analyzed. Instead of providing a constant upper bound for a loop, a symbolic formula is created using the expression representing the number of loop iterations. The formula may be evaluated later to obtain the execution time for any given input [6, 9].

We enhanced an existing timing analyzer [3, 5, 6, 9] to support generalized parametric analysis. The conversion required a three-fold process. First, a summation solver capable of summing non-constant numbers of iterations was written. Next, an advanced polynomial data structure was developed to express parametric formulas. Naturally, the polynomial data structure should not significantly slow numerical calculations. Finally, the polynomial class was integrated into the timing analyzer to replace integers in calculations. Constraint analysis – determining which paths in a loop can execute on certain iterations – can still take place with polynomial functionality.

Although these goals appear simple at first, parametric benchmarks present special problems to timing analyzers. Numerical timing analyzers receive a numerical upper bound on the maximum number of loop iterations that is either automatically determined by analyzing the program or is input by the user. Parametric analyzers receive the maximum number of loop iterations as an expression whose value is unknown at compilation (e.g. `for i = 0 .. n-1` contains *n* iterations). Because the expression must be evaluated at runtime, the loop may not execute even once (e.g. the value of *n* could be `-1`). Such uncertainty means every parametric benchmark implicitly contains control flow even when the numerical version does not.

In this paper, we describe a procedure for calculating the number of iterations of nested loops in terms of loop invariant parameters. Secondly, the existing timing tool was enhanced so parametric formulas rather than scalar quantities could be used in all calculations. Some details of our approach have been omitted for sake of brevity.

2. COMPUTING ITERATIONS

A stand-alone software package called *Emtadel* was used to calculate integral solutions for nested triangular loops. Triangular means that the induction variable of the inner loop depends on the induction variable of the outer loop. Consider the code fragment

```
for (x = 0; x < 3; ++x)              | .
  for (y = 0; y <= x; ++y)           | . .
    statement;                       | . . .
```

The dots to the right indicate the number of iterations of the inner loop for each value of x. Because the number of iterations is non-constant, the minimum, maximum, and average number of iterations must be calculated. The software package could handle any level of loop nesting and could determine if the number of loop iterations is zero even when it is not immediately apparent by examining the original loops.

For the example presented above, the equivalent summation would be

$$total\_iterations = \sum_{x=0}^{2}(\sum_{y=0}^{x}(1)) \qquad (1)$$

which equals 6. The minimum number of iterations of the inner loop is 1, the maximum is 3, and the average is 2. The average number of iterations of the inner loop is calculated by dividing the sum (6) by the total number of iterations of the outer loop (3). The concept of representing the number of loop iterations as a summation was motivated by the work of Sakellariou [8].

Averaging the number of iterations considers the possibility of the inner loop being *zero-trip* or *partially zero-trip*. A zero-trip loop derives its name from the fact that a summation whose lower bound exceeds its upper bound evaluates to zero. Hence, a zero-trip loop does not execute the loop body. A partially zero-trip loop fails to execute the loop body on some iterations. If the condition of the inner loop in the example presented above was $y < x$ (instead of $y \le x$), the inner loop would be partially zero-trip. On the first iteration of the outer loop, x would be zero and the inner loop would not execute. On the second and later iterations, the inner loop would execute. Hence, the inner loop would be partially zero-trip in the modified example. Using the average number of iterations for triangular loops achieves tight WCET bounds.

Using rational numbers in the calculations allows even more accuracy when averaging the number of iterations of triangular loop nests. In the previous example, changing the conditional expression of the outer loop to $x \le 3$ (instead of $x < 3$) would mean that the total number of iterations would be 10. Dividing by the total number of iterations of just the outer loop (4) provides the average number of iterations (2 ½). Representing the number of iterations as an integral quantity would mean that the inner loop's average number of iterations would be 3. Hence, a rational representation further tightens WCET bounds.

We use a C compiler called *vpo*, which creates a control flow information file during compilation [2]. The timing analyzer obtains loop specific information from this control flow file. This data includes the iterating (induction) variable and iteration information consisting of the minimum and maximum number of iterations (provided these are scalar quantities) or the initial, limit, and increment value of the induction variable [4]. If a parametric loop is nested within another parametric loop, the outer loop's iteration information is also reported. This allows a group of summations representing the loop structure to be generated. Emtadel handles the evaluation of the loop summations.

By means of an example, we can illustrate some of the complexities of calculating symbolic solutions when loops may be zero-trip. One example that requires a condition to be placed on the final answer is

```
for (i = 1; i <= z; i++)
  for (j = 7; j <= i; j++)
    for (k = 5; k <= i; k++)
      sum++;
```

where z is once again a function parameter. Emtadel generates a summation equivalent to the sigma notation

$$total\_iterations = \sum_{i=1}^{z}(\sum_{j=7}^{i}(\sum_{k=5}^{i}(1))) \qquad (2)$$

The innermost summation will evaluate to zero on all iterations where $i < 5$. Similarly, the middle summation will also be partially zero-trip. Emtadel is able to store such conditions on its intermediate calculations, and final answer is

$$iterations = \frac{1}{3}z^3 - \frac{9}{2}z^2 + \frac{115}{6}z - 25 \qquad (3)$$

if $z \ge 7$. If z is less than 7, the summation evaluates to zero.

This symbolic approach allows the timing analyzer to input loop variable information directly instead of having to form equations independently. The improvement replaces a significant portion of code contained within the timing analyzer and improves encapsulation. Additionally, any condition(s) placed on the final solution allows the timing analyzer itself to place conditions on the validity of its final answer.

3. ANALYSIS USING SYMBOLIC FORMULAS

The summation solver Emtadel requires a polynomial data structure to calculate and represent the symbolic solution of a summation. The timing analyzer also uses polynomials to compute parametric results. We adopt the following definition of polynomials: a polynomial contains one or more terms combined using addition. A term consists of zero or more variables each raised to a power. Each term is multiplied by either a constant (any rational number) or a *polynomial chain*. A polynomial chain allows the timing analyzer to express a parametric formula as the maximum (or minimum) of two or more polynomials. This functionality is required when the largest (or smallest) of a group of polynomials cannot be known until the values of variables are substituted into the expressions.

A polynomial chain allows both Min and Max expressions. One reason for this functionality is completeness – the polynomial class as a whole was designed to be stand-alone. But the primary reason is actually the timing analysis algorithm. Although one normally thinks of WCET as a maximum of several possible values (see Table 1), intermediate steps may require calculating a minimum. Thus, it is possible to find a Min(…) expression as part of the WCET.

As the polynomial class was written, classes from C++ standard template library – namely vectors – were used to maximize performance. The additional overhead of the polynomial class does not degrade the performance of the timing analyzer. Even timing the benchmark Matmult (see Table 1) which has fifteen timing nodes still completes in less than three seconds on a Sun Ultra 10 workstation. (The Sun Ultra 10 workstation used during the test contains an UltraSPARC II*i* processor running at 440 MHz and 256 MB of memory.) Polynomials may be simplified after each operation which means successive operations complete more quickly.

4. RESULTS

We selected several test programs to demonstrate the effectiveness of Emtadel and the enhanced timing analyzer. For sake of brevity, only a few benchmarks are shown in this paper. Parametric benchmarks are identical to scalar ones except that the limit values of loops are parameters passed into the function. The main function receives a different command line argument for each different loop limit. The argument is a left shift amount which means 2^0, 2^1, …, 2^9 are all valid numbers of maximum iterations. This approach introduces the minimum amount of extra code into the benchmark's source. We chose these numbers for convenience in testing; the timing analyzer is not limited to benchmarks which contain an unknown number of iterations which are powers of two.

Table 1 shows the results some benchmark programs. The column estimated cycles gives the execution time predicted by the timing analyzer. Observed cycles was obtained by using the integrated instruction cache and pipeline simulator which received worst case input data.

All of the benchmarks implicitly contain control flow information because they have been parameterized. The timing analyzer performs additional control flow constraint analysis as previous described in [3]. Had this analysis not been performed, the benchmark Summinmax (which contains an infeasible path) would have an additional overestimation of 5 percent. Although multiple paths exist in each timing node, only the benchmark Integral reports its final answer as the Max of two distinct polynomials. In the intermediate steps, the other benchmarks also calculated a symbolic solution using Max expression. Nonetheless, the polynomial data structure was able to determine in these other cases that one of the polynomials was always larger than the other. Hence, the smaller polynomial expression has been subsumed by the larger.

Table 1: Results for Parametric Test Programs

Program	Formula	n iterations	Observed Cycles	Estimated Cycles	Ratio
Integral	Max((153/2)n^2 – n + (193/2), (153/2)n^2 + (49/2)n + 90)	16	20,026	20,066	1.002
		128	1,256,562	1,256,602	1.000
Matmult	31n^3 + 186n^2 + 61n + 361	16	175,399	175,929	1.003
		64	8,891,095	8,892,585	1.000
Sprsin	36n^2 + 33n + 185	16	9,702	9,929	1.023
		128	592,774	594,233	1.002
Summinmax	16n + 73	16	318	329	1.035
		128	2,110	2,121	1.005

5. CONCLUSION

The contributions of this paper are twofold. First, we describe a generalized procedure for computing summations that represent the number of iterations of nested loops. This approach can handle scalar as well as multi-variate quantities and express the number of iterations in terms of loop invariant parameters. Second, we significantly enhanced an existing timing tool so that it represent both the number of iterations as well as the WCET of code segments in terms of generalized polynomial expressions rather than simply a scalar number of cycles. We then enhanced its loop analysis algorithm to take advantage of our new polynomial representation and accurate loop iteration computations. The result is that we are now able to statically bound the WCET for a larger class of benchmarks.

6. ACKNOWLEGEMENTS

This work was supported in part by NSF grants CCR-0312695, CCR-0312493, and CCR-0312531.

7. REFERENCES

[1] Aydin, H., Melhem, R., Mosse, D., and Mejia-Alvarez, P., "Power-Aware Scheduling for Periodic Real-Time Tasks," *IEEE Transactions on Computers*, *53*, *5* (May 2004), pp. 584 – 600.

[2] Benitez, M.E., and Davidson, J.W., "A Portable Global Optimizer and Linker," *Proceedings of the SIGPLAN '88 Symposium on Programming Language Design and Implementation*, June 1988, pp. 77 – 98.

[3] Healy, C., and Whalley, D., "Automatic Detection and Exploitation of Branch Constraints for Timing Analysis," *IEEE Transactions on Software Engineering*, August 2002, pp. 763 – 781.

[4] Healy, C., *Automatic Utilization of Constraints for Timing Analysis*, Ph.D. Thesis, Florida State University, 1999.

[5] Ko, L., Al-Yaqoubi, N., Healy, C., Ratliff, E., Arnold, R., Whalley, D., and Harmon, M., "Timing Constraint Specification and Analysis," *Software Practice and Experience*, January 1999, pp. 77 – 98.

[6] Mohan, S., Mueller, F., Hawkins, W., Root, M., Healy, C., and Whalley, D., "*ParaScale*: Exploiting Parametric Timing Analysis for Real-Time Schedulers and Dynamic Voltage Scaling," *Proceedings of the IEEE Real-Time Systems Symposium*, December 2005, pp. 233 – 242.

[7] Pillai, P., and Shin, K., "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proceedings of the 18th ACM symposium on Operating Systems Principles*, 2001, pp. 89 – 102.

[8] Sakellariou, R., *Symbolic Evaluation of Sums for Parallelising Compilers*, Wissenchaft & Technik Verlag, Proceedings of the 15th IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics, 1997.

[9] Vivancos, E., Healy, C., Mueller F., and Whalley, D., "Parametric Timing Analysis," *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, June 2001, pp. 88 – 93.

[10] Zinner, C., and Kubinger, W., "ROS-DMA: a DMA Double Buffering Method for Embedded Image Processing with Resource Optimized Slicing," *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2006, pp. 361 – 372.