ICT-ENERGY LETTERS

Probabilistic Resource Analysis

M. H. Kirkeby, and M. Rosendahl
Computer Science, Roskilde University, Denmark

***Abstract*— Program development has focused mainly on correctness and time efficiency, however, the size of systems has encouraged developers to include energy usage. We present a static analysis intended for software improvement; we assume a probability distribution of input is known, then given a program we instrument the program with energy usage parameters and derive a parametrized probability distribution of the programs energy usage. We define a set of imperative first-order programs for which an exact energy distribution can be derived, and demonstrate the set relevant.**

I. INTRODUCTION

Many optimizations for increased energy efficiency require probabilistic and average case analysis as part of the transformations. Wierman et al. states that "global energy consumption is affected by the average case, rather than the worst case" [1]. Also in scheduling "an accurate measurement of a tasks average-case execution time can assist in the calculation of more appropriate deadlines" [2]. For a subset of programs a precise average case execution time can be found using static analysis [3-5]. Applications of such analysis may be in improving scheduling of operations or in temperature management

We present a technique for probabilistic resource analysis where the analysis is seen as a program-to-program transformation. This means that the transformation to closed form is a source code program transformation problem and not specific to the analysis. The central idea in this paper is to use probabilistic output analysis in combination with a preprocessing phase that instruments programs with resource usage. We translate programs into a meta language program that computes the probability distribution of resource usage. This program is then analyzed and transformed with the aim of obtain a closed form expression. It is an alternative to deriving cost relations directly from the program [6, 7] or expressing costs as abstract values in a semantics for the language.

II. PROBABILITY DISTRIBUTIONS IN PROGRAMS

The result of the analysis describes the probability distribution of resource usage. We define the output probability distribution for a program *prg* in a forward manner. It is the weight or sum of all probabilities of input values where the program returns the desired value z as output.

Definition The output probability of a program, prg: $X \rightarrow Z$, (X and Z are countable sets), with a probability distribution for the input, $P_X : X \rightarrow \{r \in IR \mid 0 \leq r \leq 1\}$, the output probability distribution, $P_{prg}(z)$, is defined as:

$$P_{prg}(z) = \sum_{x \in X \wedge prg(x) = z} P_X(x) \quad (1)$$

Note that Kozen uses a similar forward definition [8], whereas Monniaux constructs the inverse and express the relationship in a backwards style [9].

Probabilistic resource analysis has similarities to worst case execution time analysis in that it depends on accurate loop/recursion bounds [10-12]. We identify a special class of loops for which exact bounds are detected by our analysis; in this class the loop's update of program variables can be described as linear expressions and the loop condition is monotone in the linear development of its variables.

Let i be a loop iteration variable, which is updated by linear expression $i=a*i+b$ over iteration variables a,b where i does not depend on neither a nor b. Let $i(n)$ denote the value of variable i at the nth loop. Let the loop-condition $test(i(n))$ be monotone in n (with the order false ≤ true) such that $test(i(n)) \leq test(i(n+1))$. The restriction on the tests along with the behavior of the variables reduces (in many cases) the execution pattern into a predictable form. Note that the linear expression for the loop iteration variable need not to occur directly, but it is enough it is derived. We call such loops *proper*.

III. RESULTS

In the following we present two programs which shows results of programs with nested proper loops parametrized input distribution of multiple variables. They are instrumented with a simple step-counter to represent the resource usage. The probability distribution computed by the output program varies in complexity; the first program calculate a single parameterized output and the second program computes a distribution converging towards a standard normal distribution. The results are presented in a reduced and readable form extracted from our implementation, where `c(test)` is a method which returns 1 if the test is true and 0, otherwise.

Matrix multiplication `mm` contains nested loops. We parametrize over the size of the matrix size n. and the wished analysis is denoted with a keyword-comment with parameter N describing n. The preprocessing phase instruments the program and converts it into a meta-language. The probabilistic output analysis computes the probability distribution, `Pmm`, describing the output distribution (out) with N as a parametrized variable.

```
// ToAnalyse: mm(_,_,_,N)

void mm(int a1[MX],int a2[MX],int a3[MX],int n){

   int i1, i2, i3, d;
                        // int step = 0;
```

```
   for(i1 = 0; i1 < n; i1++) {

      for(i2 = 0; i2 < n; i2++) {

         d = 0;
                              // step++;
         for(i3 = 0; i3 < n; i3++) {

            d = d + a1[i1*n+i3]*a2[i3*n+i2];
 // step++;
         }

         a3[i1*n+i2] = d;
                   // step++;
   }
  }

   // return step;
}
```

The nested loops create argument development functions for `i` and `step` that depend on the nested loops
. These are transformed into a linear form and removed together with the loop condition .

```
Pmm(out) =
c(3=<out/(N*N))*c(1=<N)*c(out/N*N=2+N)*1
```

The output program computes an accurate single value distribution (when specialized
with the size of the matrix N) as shown in Table 1.

N	1	2	3	4
Pmm(out) =	c(out=3)	c(out=16)	c(out=45)	c(out=96)

Tab. 1. Specializing parameter N in the computed probability distribution leads to simpler programs (e.g., when N is 3, the probability of out being 45 is 100%.).

The second program is analyzed with a distribution over input. The program sum4 adds four variables and was presented by Monniaux [9], where over approximations were applied to obtain a safe and simplified result.

```
// ToAnalyse: mm(x,y,z,w) with
Pxyzw(x,y,z,w)
// Def: P(x) = c(1=<x)*c(x=<6)*1/6

// Def: Pxyzw(x,y,z,w) = P(x)*P(y)*P(z)*P(w)
int sum4(int x, int y, int z, int w){
        //int step = 0;
    for(x; x>0;x—){    sum++;            //step++;
        for(y; y>0;y--){    sum++;       //step++;
            for(z; z>0;z--){    sum++;   //step++;
                for(w; w>0;w--){ sum++;//step++;
   }    }   }   }
return sum;                    // return step++;
}
```

The program has nested loops and in this example we use independent input variables
each uniformly distributed input from 1 to 6.

```
Psum4(out) =

c(4=<out)*c(out=<7)*

    (-12+17*out-6*out^2+out^3)/7776+

c(8=<out)*c(out=<12)*

    (-1014+169*out+6*out^2-out^3)/7776+

c(9=<out)*c(out=<12)*

    (1512-461*out+42*out^2-out^3)/3888+

c(out=13)*(265/648-5*out/216)+

    c(14=<out)*c(out=<18)*

            (-4790+923*out-54*out^2+out^3)/2592+

            c(19=<out)*c(out=<24)*

(17550-2027*out+78*out^2-out^3)/7776
```

Despite the ranges and their associated value are not symmetric, the resulting
program computes a precise and perfectly symmetric probability distribution.

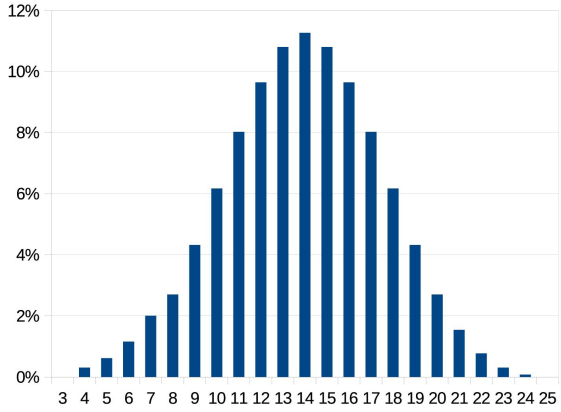

Fig. 1. The distribution as described by the output program. Here Psum4(out) is computed for out between 3 and 25.

The
differences in the choice of ranges comes (among other things) from the range
dividing rules, as they do not divide the range symmetrically. As expected from the central limit theorem of probability theory, the resulting probability program
describes a distribution that has similarities with a normal distribution.

IV. CONCLUSION

Probabilistic analysis of program has a renewed interest for analyzing programs
for energy consumptions. Numerous embedded systems and mobile applications
are limited by restricted battery life on the hardware. We have outlined the architecture of a probability resource analysis that derives a resource probability distribution for programs
given distributions of the input. We have shown some of the potential of probabilistic resource analysis and shown that
our analysis improves on related analysis in the literature.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union
Seventh Framework Programme (FP7/2007-2013) under grant agreement no 318337,
ENTRA - Whole-Systems Energy Transparency.

REFERENCES

[1] A. Wierman, L. L. H. Andrew, and A. Tang. Stochastic analysis of power-aware
scheduling. In Proceedings of Allerton Conference on Communication, Control and
Computing. Urbana-Champaign, IL, 2008.