

## Embedded Software for control

Presented by  
Betül Büyükkurt

## Presentation compiled from...

- Y. Li et al. "Performance analysis of embedded software using implicit path enumeration", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Dec. 1997, vol. 16, (no. 12): 1477-1487
- Balarin et al. "Synthesis of software programs for embedded control applications", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, (no. 6), IEEE, June 1999, p. 834-849

## Performance Analysis of Embedded Software

- ♦ SoC (System On a Chip)
  - Embedded processor, memory, peripherals and gate arrays specific integrated circuit on a single integrated circuit.
  - The main leading force to their growth is due to the migration from application specific logic to application specific code
- ♦ Main challenge with application specific code:
  - Satisfying the time constraints

## The need to know the time bounds...

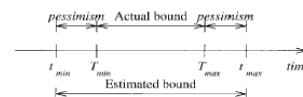
- ♦ Hard real time systems have to meet hard deadlines; therefore as system's response time is strictly bounded
- ♦ Most of the RTOS schedulers require time bounds of processes to be known in advance
- ♦ While designing systems, selection of which hardware component to use as well as the partition between hardware and software are strongly driven by the performance of software

## Problem Statement

- ♦ How to bound (lower and upper) the running time of a given program on a given processor assuming uninterrupted execution.
- ♦ The term "program" here refers to any sequence of code and does not have to include logical beginning and end.
- ♦ The running time of a program may vary according to different input data and initial machine state.

## Problem Statement – cont...

- ♦ For a given program, suppose  $T_{\min}$  and  $T_{\max}$  are the minimum and maximum of all the possible running times of the program.
- ♦  $[T_{\min}, T_{\max}]$  is called the actual bound.
- ♦ Then  $[t_{\min}, t_{\max}]$  is called the estimated bound, which must be close to the actual bound.



## Subproblems

- ♦ To predict the performance of a given piece of software on a given processor, we must:
- ♦ determine what sequence of instructions will be executed in the extreme case — this is referred to as the *program path analysis problem*;
- ♦ compute how much time it will take the system to execute that sequence — this requires a modeling of the host processor system, which is referred to as *microarchitectural modeling*.

## Challenges

- ♦ A static analysis of the code is needed, however the problem is undecidable.
- ♦ In practice, easily to bound programming styles and data structures are used, which help the problem become decidable.
  - Nodynamic data structures / i.e. pointers, dynamic arrays
  - Norecursion
  - Nounded loops

## Challenges

- ♦ Whether to do the analysis at the programming language level or the assembly level.
  - It is difficult to predict time bounds for high-level constructs independent of the context it executes
  - While applying optimizations, compiler transform programs so aggressively that high-level constructs no longer maintain indirect correspondence with the executed codes.
  - High level language phase is better to provide annotations and assembly language level is better to perform the final analysis

## Challenges

- ♦ Functionality of a program determines the path taken during execution. The challenge is whether to infer the functionality from the code or expect the programmer to provide them.
  - The amount of information that can be extracted from the code is limited.
  - It is easier for the programmer to provide any such information since he is familiar with the functionality of the code.
  - Any such information can be provided either as an annotation to the program or a separate file to a timing analyzer.
  - E.g. loop bounds, maximum execution counts of given statements if known.

## More Challenges

- ♦ Though the annotation mechanism is useful, fail to capture any information about the functional interactions between different parts of the program.
  - Proposed is that set of all possible path sequences through the program can be expressed using regular expressions.
  - Need to be explicitly specified by the programmer using a language named (IDL) Information Description Language.
  - Then the analysis algorithm determines the path sequences that can never occur, the best and the worst case paths.

## Using ILP

- ♦ **Problem:** Number of all possible feasible paths in a program is typically exponential in the size of the program.
- ♦ **Objective:** A method that tells the extreme case running times without explicitly enumerating all program paths.
- ♦ **Method:** Using integer linear programming, since ILP considers all paths implicitly in its solution; moreover the actual computation done by the solver is only solving a single linear program.

## Formulation

- Basic block: maximal sequence of instructions for which the only entry is the first instruction and the only exit is the last one
- $x_i$ : number of times a basic block is executed
- $c_i$ : the running time of basic block in the worst case
- Maximize:  $\sum_{i=1}^N c_i x_i$  wrt a set of linear constraints

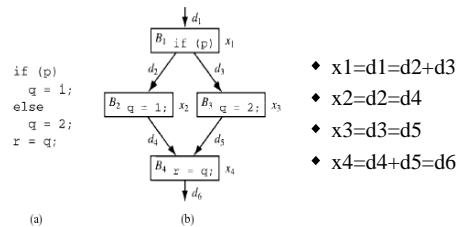
## Linear Constraints

- Helps ILP determine the values maximizing the above given formula
- Without the constraints the problem is undecidable
- There are two types of linear constraints
  - Program Structure Constraints
  - Program Functionality Constraints

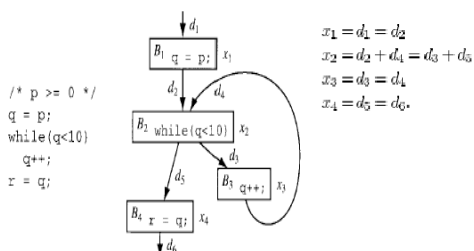
## Program Structure Constraints

- Arise from the CFG (Control Flow Graph)
- Specified as follows: At each node of the CFG, the number of times the node is executed should be equal to the sum of the control flow (edges) entering and exiting it.

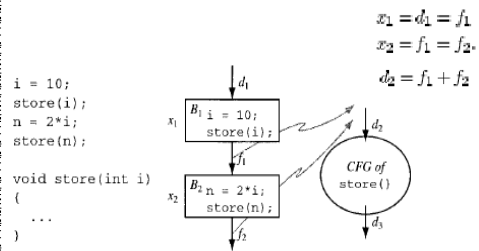
## Example



## Example - 2



## Example - Function



## Program Functionality Constraints

- ◆ These constraints describe all functionality related constraints such as loop bounds, the parts of the code that has to execute together or mutually exclusive, etc...

## Example

```

1: check_data()
2: {
3:     int i, morecheck, wrongone;
4:     morecheck = 1; i = 0; wrongone = -1;
5:     while (morecheck) {
6:         if (data[i] < 0) {
7:             wrongone = i; morecheck = 0;
8:         }
9:         else
10:            if (++i >= DATASIZE)
11:                morecheck = 0;
12:            }
13:     if (wrongone >= 0)
14:         return 0;
15:     else
16:         return 1;
17: }

```

$$1x_1 \leq x_2$$

$$x_2 \leq 10x_1.$$

$$(x_3 = 0 \ \& \ x_5 = 1) \vee (x_3 = 1 \ \& \ x_5 = 0),$$

$$(x_3 = 0 \ \& \ x_8 = 0) \vee (x_3 = 1 \ \& \ x_8 = 1),$$

$$x_3 + x_5 = 1.$$

$$x_3 = x_8.$$

## Example

```

check_data()
{ ...
x7   if (wrongone >= 0)
x8     return 0;
     else
x9     return 1;
}

task()
{ ...
f10  status = check_data();
x11  if (!status)
f12  clear_data();
...
}

```

$$f_{12} = x_8 \cdot f_{10}.$$

$$x_8 = x_8 \cdot f_{10}.$$

$$x_8 = x_8 \cdot f_{10} + x_8 \cdot f_j.$$

## Solving the Constraints

- ◆ Each of the functionality constraint sets is combined with the set of structural constraints.
- ◆ This combined constraint set is passed to the ILP solver with to be maximized.
$$\sum_{i=1}^N C_i x_i =$$
- ◆ The ILP solver solves basic block variables that provides the maximum value to the expression.
- ◆ This procedure is repeated for every functionality constraint set.
- ◆ Then the maximum overall of the collected running times is taken as the maximum running time of the program.