EE249 Discussion Session

Performance Analysis of Embedded Software Using Implicit Path Enumeration

David Ruiz Alonso
Stefano Zanella

October 26, 1999

Outline

- Introduction
- Previous Work
- Proposed Methodology
- Microarchitectural Modeling
- Implementation
- Experimental Results
- Summary
Introduction

- Embedded System => Processor running application-specific dedicated software
- “System on chip”
  - Embedded processor
  - Memory
  - Peripherals
  - Gate array application-specific logic
Introduction

- Application-specific logic
  - High Volume parts
    - Processors
    - Memories
    - FPGA
  - Speed constraints
- Application-specific software
  - Cheaper
  - Time to market

Migration driven by:
- Cost of setting up a fabrication line ($1 billion)
- Time to market
Introduction

- Basic unit of computation
  logic gate => Instruction running on a processor

- Research towards solving analysis
  and optimization problems for
  embedded software

Objective

- Determine the extreme (best and worst)
  case bounds on the running time of a
  given program on a given processor
  - Hard real-time systems
  - Schedulers in real time Operating Systems
  - Partition Hardware/Software
  - Selection of Hardware
Problem statement

- “Bound the running time of a given program on a given processor assuming uninterrupted execution”
  - “Program”: any sequence of code
  - “Processor”: System

Problem statement

- Actual bound $[T_{\text{min}}, T_{\text{max}}]$
- Estimated bound $[t_{\text{min}}, t_{\text{max}}]$

![Diagram]

- Actual bound: $[T_{\text{min}}, T_{\text{max}}]$
- Estimated bound: $[t_{\text{min}}, t_{\text{max}}]$

Pessimism: $[t_{\text{min}}, t_{\text{max}}]$ vs. $[T_{\text{min}}, T_{\text{max}}]$
Subproblems

- Prediction of extreme case performance:
  
  Program path analysis problem
  (sequence of instruction)
  Micro-architectural modeling
  (time to execute the sequence)

Bounds Needed

- Static analysis of code => undecidable
- Use easily bounded algorithms and data structures
  - Absence of dynamic data structures
  - Absence of recursion
  - Bounded loops

(Puschner and Koza, Kligerman and Stoyenko)
Restrictions Trough

- Specific language constructs
  - Predictability
  - Cost of developing a new programming language
- Programmer annotations on conventional programs
  - Performance
- Mechanism external to the language

Mok et al, Puschner and Koza, Park and Shaw

Previous Work
Analysis at...

- Programming language level
  - Shaw: time bounds for each high level language
  - Difficult to predict bounds independent of context, compiler and target processor
    => augmented solution by providing limited interaction with assembly code

- Assembly language level
  Mok et al: Functional information at the high level
    Analysis at the assembly language program

Information Needed

- The functionality of a program determines actual paths taken during its execution
- Functionality provided by the programmer
- Basic information
  - Loop bounds
  - Maximum execution counts of a given statement within a given scope
The set of all possible path sequences through the program can be expressed as regular expressions. Information Description Language (IDL) (Park and Shaw)

- Information about how different parts of the program interact
- Specify most traces that can actually happen
- Used to eliminate from analysis not feasible paths
- Explicit analysis of feasible paths to determine best and worst case paths

Regular expressions are not amenable for specification by programmers. Give up of full generality for ease of use and analysis. Complexity of computation of negation and intersection for regular expressions => approximate solutions. Need to explicit examine feasible paths => exponential blowup of paths.
Exponential Blowup

- Exponential blowup of paths

```c
for (i=0 ; i<100 ; i++) {
    if (rand() > 0.5)
        j++;
    else
        k++;
}
```

- $2^{100}$ possible paths

Proposed Methodology
Methodology Highlights

- Implicitly considers all feasible paths
- Convert the problem of determining the bounds to one solving a set of integer linear programming (ILP) problems

Solving the Problem

- Problem solved using Integer Linear Programming techniques (ILP)
- Objectives
  - Determine worst case running times
  - Worst case path non strictly necessary (?)
- Best case can be obtained in the same way
Problem Definition

- $B_i$: building blocks: one entry point and one exit point
- $x_i$: # of times $B_i$ is executed
- $c_i$: $B_i$’s running time
- Total running time: $t = \sum_i c_i x_i$
- $x_i \rightarrow$ without additional constraints

Bounding the Running Time

- Without bounds the problem is undecidable
- Must impose some restrictions
  - Limit the number of iterations in loops
    $\Rightarrow x_i$ upper bounded
  - Bounds are typically very loose
- Must consider:
  - Structure
  - Functionality
Example - Structure

If check then action 1 ($A_1$) else action 2 ($A_2$) endif

Total $c_1 + c_2 + c_3$

Example - Functionality

For $i=1:k$
If OK then block 3
else block 4 endif
end

Total $k \times [c_1 + c_2 + \max(c_3, c_4) + c_5]$
October 26, 1999

Control Flow Graph

ILP Formulation

- Both the cost function and the constraints can be expressed in a linear fashion
- Structural constraints
  - CFG & equations
- Functional constraints
  - Inequalities
  - Can be user defined
- Problem: find the set \( X = \{ x_i \in \text{CFG} \} \) such that:
  \[ t = \sum x_i c_i x_i \]
  is maximal \( \Rightarrow \text{ILP problem} \)
Setting the Constraints

\[
\begin{align*}
  x_1 &= d_1 = d_2 \\
  x_2 &= d_2 + d_4 = d_3 + d_5 \\
  x_3 &= d_3 = d_4 \\
  x_4 &= f_1 = d_5 \\
  d_6 &= \# \text{func( ) calls} \\
  d_6 &= 1 \text{(in this case)} \\
  d_7 &= d_6 \\
  x_1 &\leq x_2 \\
  x_2 &\leq n x_1 
\end{align*}
\]

More Constraints

\[
\begin{align*}
  (x_1 = x_3 = 1) &\land (x_2 = x_4 = 0) | \\
  (x_1 = x_3 = 0) &\land (x_2 = x_4 = 1)
\end{align*}
\]

disjunction operator

conjunction operator
Solving the Constraints

- The constraints are passed to an ILP solver
- An ILP problem must be solved for every combination of disjoint constraints
  \[ c_1 \mid c_2 \mid c_3 \quad c_4 \mid c_5 \rightarrow 6 \text{ problems!!!!} \]
- In practice:
  - Transform the disjoint constraints in a set of equations
  - Detect combinations with empty solutions in advance

Example

\[(x_1 = 1) \& (x_2 = 0) \mid (x_1 = 0) \& (x_2 = 1)\]

\[x_1 + x_2 = 1\]

It works because \(x_1\) and \(x_2\) must be integer!!
Detecting Null Sets

If we find a constraints:

\[ x_a \leq b \]

and later we find:

\[ x_a \geq b' \]

and

\[ b \leq b' \]

\[ \rightarrow \]

NO SOLUTION!!

Complexity

- ILP is NP-complete
- In certain cases polynomial complexity is achieved
- Structural constraints + IDL-like functional constraints
- Full set of constraints results in an ILP problem
- Authors claim that in practice this never happens

LP problem (poly time)
Microarchitectural Modeling

- Very simple model used
- Bounds derived from hardware model
- The max (min) running time of a block is the sum of the max (min) times of the instructions contained in it
- Cache always hitting (missing) is assumed for best (worst) case analysis
- Microarchitectural modeling might be a good idea for a start-up....

Cinderella

Source file

Executable file

Functionality constraints

Cinderella

Estimated bounds, Block counts

User

Annotated files
Experimental results

- Estimated running time of the program
  \[ \sum_{i=1}^{N} C_i x_i \]

- Pessimism in \( x_i \) => Insufficient functional constraints
  Experiment 1

- Pessimism in \( c_i \) => Micro-architectural model
  Experiment 2

- Target processor: Intel i960KB
Experiment 1 – Path analysis accuracy

- Calculated bound
  \[
  \left[ \sum_{i=1}^{N} c_i^{\text{best}} \times \text{measured \ block \ count}^{\text{best \ data}} \right] \\
  \left[ \sum_{i=1}^{N} c_i^{\text{worst}} \times \text{measured \ block \ count}^{\text{worst \ data}} \right]
  \]

- Estimated bound by Cinderella
  \[
  \left[ \sum_{i=1}^{N} c_i^{\text{best}} \times \text{Estimated \ block \ count}^{\text{best \ data}} \right] \\
  \left[ \sum_{i=1}^{N} c_i^{\text{worst}} \times \text{Estimated \ block \ count}^{\text{worst \ data}} \right]
  \]

- Input Data affect running time at:
  - Micro-architectural level
  - Program path level

- Impossible to to run the program for all feasible input data

- Pessimism
  \[
  \text{lower} = \frac{\text{Mea. lower} - \text{Est. lower}}{\text{Mea. lower}} \\
  \text{upper} = \frac{\text{Mea. upper} - \text{Est. upper}}{\text{Mea. upper}}
  \]
Experiment 1 – Path analysis accuracy -

<table>
<thead>
<tr>
<th>Program</th>
<th>Constraint Set</th>
<th>Pessimism</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower</td>
</tr>
<tr>
<td>check_data</td>
<td>4 =&gt; 2</td>
<td>0.00</td>
</tr>
<tr>
<td>circle</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>des</td>
<td>2</td>
<td>0.01</td>
</tr>
<tr>
<td>hry</td>
<td>8 =&gt; 3</td>
<td>0.00</td>
</tr>
<tr>
<td>djpeg</td>
<td>1</td>
<td>0.02</td>
</tr>
<tr>
<td>fdct</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>Ft</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>line</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>matcnt</td>
<td>1</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Experiment 2 – Actual Running Times -

- **Best case**
  - All instruction fetches result in cache hits

- **Worst case**
  - All instruction fetches result in cache misses
Experiment 2 – Actual Running Times -

<table>
<thead>
<tr>
<th>Program</th>
<th>Constraint Set</th>
<th>Pessimism</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower</td>
</tr>
<tr>
<td>check_data</td>
<td>4 =&gt; 2</td>
<td>0.00</td>
</tr>
<tr>
<td>circle</td>
<td>1</td>
<td>0.26</td>
</tr>
<tr>
<td>des</td>
<td>2</td>
<td>0.34</td>
</tr>
<tr>
<td>hry</td>
<td>8 =&gt; 3</td>
<td>0.45</td>
</tr>
<tr>
<td>djpeg</td>
<td>1</td>
<td>0.15</td>
</tr>
<tr>
<td>fdct</td>
<td>1</td>
<td>0.27</td>
</tr>
<tr>
<td>Ft</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>line</td>
<td>1</td>
<td>0.59</td>
</tr>
<tr>
<td>matcnt</td>
<td>1</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Summary

- An efficient method for embedded software performance has been presented
- ILP is used in order to avoid explicit path enumeration
- Experimental result show that the approach is very efficient
- Open issues:
  - Cache modeling
  - Usage of dataflow analysis to automatically derive some constraints