Precise and Efficient Parametric Path Analysis

Ernst Althaus, Sebastian Altmeyer, Rouven Naujoks

Johannes Gutenberg-Universität Mainz
Saarland University
Max-Planck-Institut für Informatik

LCTES 2011, Chicago
1 Parametric Timing Analysis
   Motivation
   Toolchain
   Path Analysis

2 Singleton Loop Model
   What is a Singleton Loop?
   Exploit the Singleton Loop Model
   Runtime
   Beyond Single Loops

3 Evaluation
   Structure of the Benchmarks
   Performance Evaluation
   Precision

4 Conclusions
Parametric Timing Analysis – Why?

- timing analysis essential for hard real-time systems
- many systems depend on input parameters (operating system schedulers, etc.)

- only two possible solutions:
  1. assume upper bounds on the unknown parameters ⇒ highly overapproximated execution-time bound
  2. restart the analysis for all parameter assignments ⇒ very high analysis time

- parametric timing analysis delivers timing formula instead of a numeric value
Parametric Timing Analysis – How?

CFG Reconstruction extracts the control flow graph from the executable.

Value/Loop Analysis determines values for registers and memory accesses determines loop bounds and parametric loop bound expressions.

Pipeline Analysis derives bounds on the execution times $T(v_i)$ of all basic blocks.

Path Analysis combines execution times of basic blocks and loop bounds to determine longest execution path.

Framework according to [1].
Path Analysis; Longest Paths via ILP

\[
\max \sum_i \sum_{n_j \in \text{inc}(v_i)} T(v_i) n_j
\]

- \( n_1 = 1 \)
- \( n_1 = n_2 + n_3 \)
- \( n_2 + n_5 = n_4 + n_6 \)
- \( n_4 = n_5 \)
- \( n_3 + n_6 = 1 \)
- \( n_4 \leq b_l \cdot n_2 \)

- \textit{Implicit path enumeration} (IPET [4])
- Control flow graph and the loop bounds are transformed into \textit{flow constraints}.
- Upper bounds for the execution times used as weights.

Althaus, Altmeyer, Naujoks

Precise and Efficient Parametric Path Analysis
Parametric Path Analysis; Longest Paths via ILP

\[
\max \sum_i \sum_{n_j \in inc(v_i)} T(v_i)n_j
\]

\[
\begin{align*}
n_1 &= 1 \\
n_1 &= n_2 + n_3 \\
n_2 + n_5 &= n_4 + n_6 \\
n_4 &= n_5 \\
n_3 + n_6 &= 1 \\
n_4 &\leq b_1 n_2 \quad b_1 \cdot c
\end{align*}
\]

- Non-linear inequalities \( \Rightarrow \) need for approximation
Parametric Path Analysis; Longest Paths via ILP

\[
\max \sum_i \sum_{j \in \text{inc}(v_i)} T(v_i) n_j
\]

\[
\begin{align*}
    n_1 &= 1 \\
    n_1 &= n_2 + n_3 \\
    n_2 + n_5 &= n_4 + n_6 \\
    n_4 &= n_5 \\
    n_3 + n_6 &= 1 \\
    n_4 &\leq \ b_1 \cdot n_2 \ \ b_1 \cdot c
\end{align*}
\]

- Non-linear inequalities \(\Rightarrow\) need for approximation
- Need to solve an ILP/parametric PIP [3]
- slow and imprecise in case of parametric ILP
Determing Longest Paths in Control Flow Graphs

Problem is NP-hard in general
Determining Longest Paths in Control Flow Graphs

Problem is NP-hard in general

But: may be solved efficiently for restricted graphs
    ⇒ Singleton-Loop Model
What is a Singleton Loop?

Idea: Code for hard real-time systems often well structured.

- A loop in a CFG is a *strongly connected component* (SCC).
- Structured loops (no Gotos etc.) have a *single entry node*.

A *singleton loop* is a SCC with exactly one entry node.

A *singleton loop graph* is a CFG that contains only singleton loops.
Assume we know for each loop (by recursion):
Assume we know for each loop (by recursion):

- Longest paths from its entry node to its portal nodes.
• Assume we know for each loop (by recursion):
  • Longest paths from its entry node to its portal nodes.
• Contract loop to artificial node $N$.
  • set weight of incident edges appropriately
    \[ w_1 := lps(v_1, v_4) + w(v_4, v_6), \]
    \[ w_2 := lps(v_1, v_5) + w(v_5, v_6). \]
Assume we know for each loop (by recursion):
- Longest paths from its entry node to its portal nodes.
- Contract loop to artificial node $N$.
  - set weight of incident edges appropriately
    \[ w_1 := lps(v_1, v_4) + w(v_4, v_6), \]
    \[ w_2 := lps(v_1, v_5) + w(v_5, v_6) \]
- Left with a directed acyclic graph.
  - Longest Path Computation in polynomial time.
The Singleton-Loop Model - How to recurse

- Given a loop $L$, with loop bound $b_L$.
- Recall: want to determine LPs from entry node to portal nodes.

 Geoffrey Alonso, Altmeyer, Naujoks Precise and Efficient Parametric Path Analysis
The Singleton-Loop Model - How to recurse

- Given a loop $L$, with loop bound $b_L$.
- Recall: want to determine LPs from entry node to portal nodes
- Replace entry node $v_1$ by
  - two nodes $v_1^{in}, v_1^{out}$ with
  - in- and outgoing edges of $v_1$ assigned accordingly
The Singleton-Loop Model - How to recurse

- Given a loop $L$, with loop bound $b_L$.
- Recall: want to determine LPs from entry node to portal nodes.
- Replace entry node $v_1$ by:
  - two nodes $v_1^{in}$, $v_1^{out}$ with
  - in- and outgoing edges of $v_1$ assigned accordingly.
- Recurse algorithm on this new graph.
  - we know $LP(v_1^{out}, v_i)$ and $LP(v_1^{out}, v_1^{in})$. 
The Singleton-Loop Model - How to recurse

- Given a loop $L$, with loop bound $b_L$.
- Recall: want to determine LPs from entry node to portal nodes.
- Replace entry node $v_1$ by:
  - two nodes $v_1^{in}, v_1^{out}$ with
  - in- and outgoing edges of $v_1$ assigned accordingly
- Recurse algorithm on this new graph:
  - we know $LP(v_1^{out}, v_i)$ and $LP(v_1^{out}, v_1^{in})$
  - $lps(v_1, p_i) := (b_L - 1) \cdot lps(v_1^{out}, v_1^{in}) + lps(v_1^{out}, p_i)$
Runtime Properties

Worst Case Running Times

- numeric bounds: $O(|V||E|)$
- symbolic bounds: $O(|V||E| + |V|^2 \cdot x \cdot s(x))$ where
  - $x$ is the number of symbolic bounds
  - $s(x)$ is the output size

Output Size

In the worst case:

$$2^{2x-1} \leq s(x) \leq 2^{2x}$$

Output Sensitivity

The algorithm is polynomial output sensitive, i.e. its running time is polynomial in the input size and in the output size.
Beyond the Singleton-Loop Model

What happens, if CFG has non-singleton loops?

Each CFG can be transformed into a Singleton Loop Graph.

Disadvantage:
Comes at the cost of increased running time!
(e.g. symbolic bounds can be doubled!)

Althaus, Altmeyer, Naujoks
Precise and Efficient Parametric Path Analysis
Beyond the Singleton-Loop Model

What happens, if CFG has non-singleton loops?

Convert the CFG!

Each CFG can be transformed into an Singleton Loop Graph
Beyond the Singleton-Loop Model

What happens, if CFG has non-singleton loops?

Convert the CFG!

Each CFG can be transformed into a Singleton Loop Graph

Disadvantage: Comes at the cost of increased running time!
(e.g. symbolic bounds can be doubled!)
How useful is the new approach in practise?

Depends on:

- How well does the Singleton-Loop Model fits real CFGs?
- How does the Singleton-Loop Approach perform?
- How much precision is gained?

Testsetting:

- Benchmarks from Mälardalen WCET benchmark suite.
- Compiled via gcc to the ARM7 processor.
- Analyzed on an Intel Core2Duo, 2GHz, 2 GB Ram with Ubuntu 9.10.
Number of Singleton Loops

- Only 8 of 33 test-cases exhibit non-singleton loops (adpcm, cnt, compress, duff, matmult, ndes, ns, qsort-exam).
- Only in one case (compress) a higher number of loop-duplications (65) is needed (all others < 10).

Deeply nested loops, unstructured code segments, calls to external library functions causes non-singleton loops.
Compare performance to?

Numerica Path Analysis

- ILP Formulation with Ip_solve (free Ip-solver)
- ILP Formulation with CPLEX (commercial Ip-solver)

Parametric Path Analysis

- Parametric ILP Formulation with PIP [3] (free parametric Ip-solver)
- Parametric timing analysis by Bygde and Lisper [5, 2] resorts to C-level, not to binary level, uses a polyhedron approach; direct comparison not possible
Performance Evaluation – Test-Cases are very small

Testcases from Mälardalen WCET benchmark suite are very small (all are solved in less than 1 second by all approaches)

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (in Byte)</th>
<th>Singleton Graph</th>
<th># duplicated loops</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C-File</td>
<td>Exec</td>
<td></td>
</tr>
<tr>
<td>s-graph-1</td>
<td>208273</td>
<td>235222</td>
<td>yes</td>
</tr>
<tr>
<td>s-graph-2</td>
<td>468944</td>
<td>292305</td>
<td>yes</td>
</tr>
<tr>
<td>s-graph-3</td>
<td>702670</td>
<td>386961</td>
<td>yes</td>
</tr>
<tr>
<td>s-graph-4</td>
<td>936396</td>
<td>481609</td>
<td>yes</td>
</tr>
<tr>
<td>s-graph-5</td>
<td>670452</td>
<td>284593</td>
<td>yes</td>
</tr>
<tr>
<td>ns-graph-1</td>
<td>90274</td>
<td>215433</td>
<td>no</td>
</tr>
<tr>
<td>ns-graph-2</td>
<td>315562</td>
<td>247443</td>
<td>no</td>
</tr>
<tr>
<td>ns-graph-3</td>
<td>766144</td>
<td>426427</td>
<td>no</td>
</tr>
<tr>
<td>ns-graph-4</td>
<td>990502</td>
<td>520579</td>
<td>no</td>
</tr>
<tr>
<td>ns-graph-5</td>
<td>979908</td>
<td>518338</td>
<td>no</td>
</tr>
<tr>
<td>ns-graph-6</td>
<td>942084</td>
<td>502580</td>
<td>no</td>
</tr>
</tbody>
</table>

Larger benchmarks created by combining and duplicating original test-cases from the benchmark suite (s-graph-X are singleton loop graphs, ns-graph-X non-singleton loop graphs)
## Performance Evaluation, Numeric

<table>
<thead>
<tr>
<th>Name</th>
<th>Runtime (s)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SingletonLoop</td>
<td>lp_solve</td>
<td>CPLEX</td>
</tr>
<tr>
<td>nsichneau</td>
<td>0.02</td>
<td>0.86</td>
<td>0.05</td>
</tr>
<tr>
<td>s-graph-1</td>
<td>0.03</td>
<td>3.46</td>
<td>0.08</td>
</tr>
<tr>
<td>s-graph-2</td>
<td>0.05</td>
<td>13.69</td>
<td>0.08</td>
</tr>
<tr>
<td>s-graph-3</td>
<td>0.08</td>
<td>30.85</td>
<td>0.11</td>
</tr>
<tr>
<td>s-graph-4</td>
<td>0.11</td>
<td>57.31</td>
<td>0.18</td>
</tr>
<tr>
<td>s-graph-5</td>
<td>0.11</td>
<td>108.8</td>
<td>0.13</td>
</tr>
<tr>
<td>adpcm</td>
<td>0.04</td>
<td>0.07</td>
<td>0.02</td>
</tr>
<tr>
<td>compress</td>
<td>0.3</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>state mate</td>
<td>0.05</td>
<td>0.3</td>
<td>0.04</td>
</tr>
<tr>
<td>ns-graph-1</td>
<td>0.97</td>
<td>4.5</td>
<td>0.04</td>
</tr>
<tr>
<td>ns-graph-2</td>
<td>0.95</td>
<td>14.58</td>
<td>0.05</td>
</tr>
<tr>
<td>ns-graph-3</td>
<td>1.01</td>
<td>48.13</td>
<td>0.12</td>
</tr>
<tr>
<td>ns-graph-4</td>
<td>0.14</td>
<td>92.1</td>
<td>0.11</td>
</tr>
<tr>
<td>ns-graph-5</td>
<td>0.16</td>
<td>113.3</td>
<td>0.12</td>
</tr>
<tr>
<td>ns-graph-6</td>
<td>0.64</td>
<td>65.9</td>
<td>0.17</td>
</tr>
</tbody>
</table>
Performance Evaluation, Parametric

<table>
<thead>
<tr>
<th>Name</th>
<th>Runtime # of parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>s-graph-1</td>
<td>0.03</td>
</tr>
<tr>
<td>s-graph-2</td>
<td>0.05</td>
</tr>
<tr>
<td>s-graph-3</td>
<td>0.08</td>
</tr>
<tr>
<td>s-graph-4</td>
<td>0.11</td>
</tr>
<tr>
<td>s-graph-5</td>
<td>0.11</td>
</tr>
<tr>
<td>ns-graph-1</td>
<td>0.97</td>
</tr>
<tr>
<td>ns-graph-2</td>
<td>0.95</td>
</tr>
<tr>
<td>ns-graph-3</td>
<td>1.01</td>
</tr>
<tr>
<td>ns-graph-4</td>
<td>0.14</td>
</tr>
<tr>
<td>ns-graph-5</td>
<td>0.16</td>
</tr>
<tr>
<td>ns-graph-6</td>
<td>0.64</td>
</tr>
</tbody>
</table>

Measurements only of singleton loop method
All other approaches fail to solve these test-cases
(PIP and Bygde’s approach [2] handle at most two parameters)
Evaluation: Precision of the Parametric Formulas

\[ Time_{\text{PIP}}(n) = 156n^2 + 674n + 1186 \]
\[ Time_{\text{Singleton}}(n) = 131n^2 + 71n + 1185 \]

\[ Time_{\text{PIP}}(n) = \begin{cases} 
386n^3 + 782n^2 \\
+ 790n + 643 & \text{if } n > 1 \\
2992 & \text{if } n \leq 1 
\end{cases} \]
\[ Time_{\text{Singleton}}(n) = 111n^3 + 164n^2 + 845n + 793 \]

- Singleton Loop Method is precise
- PIP suffers from imprecision due to loop bound transformation.
- Bygde’s approach is precise in most, but not in all cases
Conclusions

Singleton Loop Graphs are restricted CFG that enable computation of

- numeric timing bound in polynomial time,
- parametric timing bound in output-polynomial time (significant improvment over former methods), and
- precise parametric timing bounds.

All CFGs can be transformed to singleton loop graphs (at the cost of performance loss).

Evaluation showed that

- most benchmarks fit the singleton loop model,
- singleton loop approach can compete with CPLEX,
- enable fast and precise computation of parametric timing bounds.
Thanks for your attention.
References

S. Altmeyer, C. Hümbert, B. Lisper, and R. Wilhelm.
Parametric timing analysis for complex architectures.
In *RTCSA’08*, 2008.

S. Bygde, A. Ermedahl, and B. Lisper.
An efficient algorithm for parametric WCET calculation.
In *RTCSA’09*, 2009.

P. Feautrier.
The parametric integer programming’s home http://www.piplib.org.

Y.-T. S. Li and S. Malik.
Performance analysis of embedded software using implicit path enumeration.
In *DAC ’95*.

B. Lisper.
Fully automatic, parametric worst-case execution time analysis.
In *WCET 03*.
<table>
<thead>
<tr>
<th>Name</th>
<th>Size (in Byte)</th>
<th>Singleton</th>
<th># duplicated loops</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C-File</td>
<td>Exec</td>
<td>Graph</td>
</tr>
<tr>
<td>adpcm</td>
<td>26582</td>
<td>156759</td>
<td>no</td>
</tr>
<tr>
<td>bs</td>
<td>4248</td>
<td>144447</td>
<td>yes</td>
</tr>
<tr>
<td>bs100</td>
<td>2779</td>
<td>144629</td>
<td>yes</td>
</tr>
<tr>
<td>cnt</td>
<td>2880</td>
<td>149801</td>
<td>yes</td>
</tr>
<tr>
<td>compress</td>
<td>13411</td>
<td>149804</td>
<td>no</td>
</tr>
<tr>
<td>cover</td>
<td>5026</td>
<td>148301</td>
<td>yes</td>
</tr>
<tr>
<td>crc</td>
<td>5168</td>
<td>145615</td>
<td>yes</td>
</tr>
<tr>
<td>duff</td>
<td>2374</td>
<td>144739</td>
<td>no</td>
</tr>
<tr>
<td>edn</td>
<td>10563</td>
<td>150682</td>
<td>yes</td>
</tr>
<tr>
<td>expint</td>
<td>4288</td>
<td>145867</td>
<td>yes</td>
</tr>
<tr>
<td>fac</td>
<td>426</td>
<td>144148</td>
<td>yes</td>
</tr>
<tr>
<td>fdct</td>
<td>8863</td>
<td>147128</td>
<td>yes</td>
</tr>
<tr>
<td>fft1</td>
<td>6244</td>
<td>153303</td>
<td>yes</td>
</tr>
<tr>
<td>fibcall</td>
<td>3499</td>
<td>144152</td>
<td>yes</td>
</tr>
<tr>
<td>fir</td>
<td>11965</td>
<td>151589</td>
<td>yes</td>
</tr>
<tr>
<td>insertsort</td>
<td>3892</td>
<td>144305</td>
<td>yes</td>
</tr>
<tr>
<td>jannecomplex</td>
<td>1564</td>
<td>144242</td>
<td>yes</td>
</tr>
<tr>
<td>jfdctint</td>
<td>16028</td>
<td>146858</td>
<td>yes</td>
</tr>
<tr>
<td>lcdnum</td>
<td>1678</td>
<td>144509</td>
<td>yes</td>
</tr>
<tr>
<td>lms</td>
<td>7720</td>
<td>157868</td>
<td>yes</td>
</tr>
<tr>
<td>ludpcm</td>
<td>5160</td>
<td>151848</td>
<td>yes</td>
</tr>
<tr>
<td>matmult</td>
<td>3737</td>
<td>145083</td>
<td>no</td>
</tr>
<tr>
<td>minver</td>
<td>5805</td>
<td>152845</td>
<td>yes</td>
</tr>
<tr>
<td>ndes</td>
<td>7345</td>
<td>148689</td>
<td>no</td>
</tr>
<tr>
<td>ns</td>
<td>10436</td>
<td>149567</td>
<td>no</td>
</tr>
<tr>
<td>nsichneu</td>
<td>118351</td>
<td>176240</td>
<td>yes</td>
</tr>
<tr>
<td>prime</td>
<td>904</td>
<td>144538</td>
<td>yes</td>
</tr>
<tr>
<td>qsort-exam</td>
<td>4535</td>
<td>146468</td>
<td>no</td>
</tr>
<tr>
<td>qurt</td>
<td>4898</td>
<td>151214</td>
<td>yes</td>
</tr>
<tr>
<td>recursion</td>
<td>620</td>
<td>144341</td>
<td>yes</td>
</tr>
<tr>
<td>select</td>
<td>4494</td>
<td>146283</td>
<td>yes</td>
</tr>
<tr>
<td>sqrt</td>
<td>3567</td>
<td>154282</td>
<td>yes</td>
</tr>
</tbody>
</table>
Appendix: Imprecision of Parametric ILP Approach

- Only one loop taken in actual execution
- parametric ILP needs to upper bound entry node: both loops are part of the WCET Path