Worst-Case Timing Estimation and Architecture Exploration in Early Design Phases
Early Estimation Problem
TimingExplorer

- Goal: assist in the exploration of alternative system configurations

- Design:
  - based on aiT
  - parameterizable core

- Requirements: (representative) source code
void Task (void) {
    variable++;  // Increment variable
    function();  // Call function
    while (next)  // Repeat until next becomes false
    {
        do this;  // Perform action
        next--;  // Decrement next
    }
    terminate();  // Terminate task
}
void Task (void)
{
    variable++;
    function();
    while (next)
    {
        do this;
        next--;
    }
    terminate();
}
From Source to Executable

- Compilation and linking are user’s responsibility
- Compiler effects
  - Production compiler
  - Standard compiler
aiT

Application Code

```c
void Task (void)
{
    variable++;
    function();
    next++;
    if (next)
        do this;
    terminate();
}
```

Software Characteristics

- loop "._codebook" + 1 loop exactly 16 end;
- recursion "._fac" max 6;
- SNIPPET "printf" IS NOT ANALYZED AND TAKES MAX 333 CYCLES;

Compiler

Linker

Executable (*.elf / *.out)

Entry Point

aiT

WCET
void Task (void) {
    variable++;
    function();
    next++;  
    if (next) do this;
    terminate()
}

Application Code

Compiler Linker

Executable (*.elf / *.out)

TimingExplorer

Architecture Parameterization

Software Characteristics

cache instruction
set-count = 128, assoc = 4, line-size = 32,
policy = LRU, may = empty
and data
set-count = 128, assoc = 4, line-size = 32,
policy = LRU, may = empty;

loop "_.codebook" + 1 loop exactly 16 end;
recursion "_.fac" max 6;
SNIPPET "printf" IS NOT ANALYZED
AND TAKES MAX 333 CYCLES;

Entry Point

- WCET Estimate
- Architecture Evaluation
Cache Specification

- Size, line size, replacement policy, associativity
- Cache access behaviour:
  - Default strategy for unknown accesses – hit/miss
  - Turn off cache analysis and treat all accesses as hit/miss/unknown

- Completely parameterizable memory map
Pipeline Analysis

➢ Use local worst case

Goal

▪ Reduce computation resource usage
▪ Speed up the analysis
Source-code Analyses

- Goal: reduce the need for manual annotation
- May introduce imprecision:
  
  ```cpp
  for (int i=0; i<100; i++)
  ...
  ```

- Analyses: function pointer resolution, loop bounds
  - Framework: SATIrE, LLNL-ROSE, PAG
Future work

- Case study
- Integration in a system-level analysis
  - XML Timing Cookies
Source-code Analyses

- Goal: reduce the need for manual annotation
- May introduce imprecision:
  ```c
  for (int i=0; i<100; i++)
  ...
  ```
- Analyses: function pointer resolution, loop bounds
  - Framework: SATIrE, LLNL-ROSE, PAG
Function Pointer Resolution

- Compute a points-to set for each function pointer variable
- Handles complex structures
- Output: annotations
## Test benchmark

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>Lines of code</th>
<th>Indirect calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>diction</td>
<td>0.7</td>
<td>2 037</td>
<td>3</td>
</tr>
<tr>
<td>grep</td>
<td>2.0</td>
<td>12 417</td>
<td>3</td>
</tr>
<tr>
<td>gzip</td>
<td>1.2.4</td>
<td>8 163</td>
<td>4</td>
</tr>
<tr>
<td>sim6890</td>
<td>0.1</td>
<td>3 290</td>
<td>97</td>
</tr>
</tbody>
</table>
Loop-bound Analysis

- Express the upper loop bound as a parametric formula
- One symbolic formula per loop
  - value analysis => concrete bounds
Loop-bound analysis results

- Detects and assigns a bound to all loops
- Precision (Mälardalen benchmark suite; provided precise results of the value analysis)
  - > SWEET
  - ≥ oRange
- Automatically generated code
- Effective for typical counter loops