Approximation of the Worst-Case Execution Time Using Structural Analysis

Matteo Corti and Thomas Gross

ETH Zürich
Goal

- Worst-case execution time estimation of soft-real time Java applications.
- We focus on semantic analysis:
  - compute a tight bound on the max and min number of iterations for every block
  - consider different path frequencies inside loops
  - avoid path enumeration
Outline

• Goal
• Loop bounds
• Block bounds
• Complexity and related work
• Testing environment
• Results
• Concluding remarks
System’s overview

bytecode

partial abstract interpretation

loop header bounds

block bounds

Semantic analysis

WCET

instruction duration analysis

annotated asm
Java

• Whole program analysis.
• Variable type based analysis to resolve polymorphism.
• We consider only local integer variables for the loop analysis.

• Our block iterations bounding technique is language independent.
System’s overview

- Bytecode
- Partial abstract interpretation
- Loop header bounds
- Block bounds
- WCET
- Instruction duration analysis
- Annotated asm

Semantic analysis
Partial abstract interpretation

- We perform a limited abstract interpretation pass over linear code.
- We discover some false paths (not containing cycles).
- We gather information on possible variables’ values.

```c
void foo(int i) {
    if (i > 0) {
        for(;i<10;i++) {
            bar();
        }
    }
}
```
### Partial abstract interpretation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Infeasible paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>_201_compress</td>
<td>2</td>
</tr>
<tr>
<td>_202_jess</td>
<td>3</td>
</tr>
<tr>
<td>_205_raytrace</td>
<td>7</td>
</tr>
<tr>
<td>_209_db</td>
<td>2</td>
</tr>
<tr>
<td>_213_javac</td>
<td>240</td>
</tr>
<tr>
<td>_222_mpegaudio</td>
<td>19</td>
</tr>
<tr>
<td>_228_jack</td>
<td>22</td>
</tr>
</tbody>
</table>
**Partial abstract interpretation**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Infeasible longest path</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaLayer</td>
<td>2</td>
</tr>
<tr>
<td>linpack</td>
<td>2</td>
</tr>
<tr>
<td>whetstone</td>
<td>1</td>
</tr>
</tbody>
</table>
System’s overview

bytecode

partial abstract interpretation

loop header bounds

block bounds

Semantic analysis

instruction duration analysis

annotated asm

WCET
Loop bounds

• Bounds on the loop header computed similarly to C. Healy [RTAS’98].
• We introduce noncontiguous sets of integers to easily handle equality operators.

• *Iteration branch*: a block where the conditional jump could be responsible for a loop exit.
• For each edge $e$ and iteration branch $ib$ we compute the possible number of iterations.
Loop bounds

• The bounds on the iterations of the header are safe for the whole loop.
• But: some parts of the loop could be executed less frequently:

```c
for(int i=0; i<100; i++) {
    if (i < 50) {
        A;
    } else {
        B;
    }
}
```
System’s overview

bytecode

partial abstract interpretation

loop header bounds

block bounds

Semantic analysis

WCET

instruction duration analysis

annotated asm
Basic block iterations

- The number of iterations of a block is **not** a local property (based on immediate predecessors).

```java
void foo(boolean b) {
    for(int i=0; i<10; i++) {
        if (b) {
            P0;
        } else {
            P1;
        }
    }
}
```
Basic block iterations

- The number of iterations of a block is **not** a local property (based on immediate predecessors).

```java
void foo(boolean b) {
    for(int i=0; i<20; i++) {
        if (i<10) {      P0;    } else {      P1;    }
    B;
}
```
Basic block iterations

- The number of iterations of a block is not a local property (based on immediate predecessors).

```java
void foo(boolean b) {
    int i = 0;
    if (b) {
        do {
            i++; P0;
        } while (i<10);
    } else {
        do {
            i++; P1;
        } while (i<10);
    }
    B;
}
```
Basic block iterations

• The number of iterations of a block is **not** a local property (based on immediate predecessors).
Structural analysis

- Powerful interval analysis.
- Recognizes semantic constructs.
- Useful when the source code is not available.
- **Iteratively matches the blocks with predefined patterns.**
Structural analysis

• Powerful interval analysis.
• Recognizes semantic constructs.
• Useful when the source code is not available.
• Iteratively matches the blocks with predefined patterns.
Structural analysis

Static patterns:

Dynamic patterns:

```java
if (A || (B && C)) {
    D;
} else {
    E;
} F;
```
Block iterations

- Block iterations are computed using the CFG root and the iteration branches.
- The header and the type of the biggest semantic region that includes all the predecessors of a node determine its number of iterations.

- Complete algorithm in the paper.
Example

```java
void foo(boolean b) {
    for(int i=0; i<20; i++) {
        if (i<10) {
            P0;
        } else {
            P1;
        }    B;
    }
}
```
void foo(boolean b) {
    int i = 0;
    if (b) {
        do {
            i++; P0;
        } while (i<10);
    } else {
        do {
            i++; P1;
        } while (i<10);
    }
    B;
}
Example

```java
void foo(boolean b) {
    PH;
    for (int i=0; i<10; i++) {
        L;
    }
    B;
}
```
Related work

- Automatically detected value-dependent constraints [Healy, RTAS’99]:
  - per block bounds
  - requires path enumeration (in the loop body)

- We propagate the header bounds to the blocks in quadratic time:
  - Structural analysis: $O(B^2)$
  - Loop bounds: $O(B)$
  - Block bounds: $O(B)$
Evaluation: hardware-level analysis

- The semantic analysis is platform independent.
- Evaluation: Pentium III on Linux.
- We approximate the effects of caches and pipelines:
  - we assume that the effects of an instruction fade over time.
  - caches and pipelines are analyzed locally.
- Possible sources of inaccuracies:
  - cache misses and pipeline stalls
  - **but not** the number of iterations of an instruction (conservative)
Evaluation

• **Base**: the bounds on the iterations of the loop header are used for the whole loop.

• **Enhanced**: structural analysis is used to consider different path frequencies in loop bodies.
Example 1

```c
for (i=0; i<10000; i++) {
    if (i<5000) {
        array[i] = -array[i];
    }
    if (array[i] > max) {
        max = array[i];
    }
}
```

Example 2

```c
for(i=0; i<10; i++) {
    for (j=0; j<10; j++) {
        if(j<9) {
            m[i][j] *= m[i][j];
        } else {
            for(k=0; k<9; k++) {
                m[i][j] += m[i][k];
            }
        }
    }
}
```

<table>
<thead>
<tr>
<th></th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>10'000</td>
<td>10'000</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Enhanced</td>
<td>5'000</td>
<td>10'000</td>
<td>90</td>
<td>10</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Max observed [cycles]</th>
<th>Enhanced [cycles]</th>
<th>Base [cycles]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatMult</td>
<td>2.68·10⁹</td>
<td>2.73·10⁹</td>
<td>2%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.73·10⁹</td>
</tr>
<tr>
<td>Jacobi</td>
<td>0.88·10¹⁰</td>
<td>1.08·10¹⁰</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.08·10¹⁰</td>
</tr>
<tr>
<td>JavaLayer</td>
<td>2.67·10⁹</td>
<td>1.30·10¹⁰</td>
<td>487%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.49·10⁹</td>
</tr>
<tr>
<td>SciMark</td>
<td>2.47·10¹⁰</td>
<td>1.42·10¹¹</td>
<td>579%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.12·10¹¹</td>
</tr>
<tr>
<td>_201_compress</td>
<td>9.45·10⁹</td>
<td>1.11·10¹⁰</td>
<td>117%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.76·10¹²</td>
</tr>
</tbody>
</table>
for (data = 0; data < N; data++) {

    // compress data

    if (data == 10'000) {
        // update structures
    }
}

Concluding remarks

• We tighten the bounds of basic blocks iteration considering different paths inside loop bodies.
• We do not perform path enumeration
• Tests with real applications validate the semantic analysis.
Questions?