Performance Analysis and Co-Simulation

Zebo Peng
Embedded Systems Laboratory (ESLAB)
Linköping University

Outline

- Static analysis techniques
- Worst-case execution time analysis
- Micro-architecture modeling and analysis
- Co-simulation approaches
Introduction

- Analysis and simulation techniques are essential for embedded system design:
  - To guide the design space exploration.
  - To provide feedback to the human designers.
  - To support design validation.

- Selection of an analysis/simulation technique is usually based on trade-off between efficiency and accuracy.

- For certain analysis, such as worst-case execution time analysis, it is also very important that the result is safe (i.e., correct or pessimistic).

Typical Data to be Analyzed

- Execution time and performance.
- Schedulability.
- Cost (code size, silicon area, etc.)
- Power consumption.
- Testability.
- Reliability.
- ...
Performance Metrics

- Extreme case performance
  - Worst-case execution time
  - Best-case execution time
- Average case performance
- Probabilistic performance
  - Used in soft real-time applications.
  - To accurately handle the variable execution time of tasks, which may be due to
    - Application characteristics (e.g., data dependent loops);
    - Architectural factors (e.g., cache misses);
    - External factors (e.g., network load); or
    - Insufficient knowledge.
  - To guarantee a high probability of meeting timing constraints.

Worst-Case Execution Time Analysis

- Simply measuring the execution time of a task for a given input is not safe.
  - It is impossible to prove that the conditions leading to maximum execution time are taken into account.
- Processor components like caches and pipelines complicate the task of determining the WCET considerably.
  - The execution time of a single instruction may depend on the execution history of many other instructions.
- Switching off caches to simplify WCET prediction can lead to severe performance degradation.
  - e.g., a factor of up to 30 for PowerPC 604.
Simulation-based Techniques

- Software — Running the compiled program on the simulated target architecture.
- Hardware — Building a simulation model of the hardware and simulating.
- A very large number of inputs should usually be used in order to get good results.
- Only practical for average and probabilistic execution time estimation.
- It is difficult to use when individual programs are not running in isolation.

Profiling

- A simulation-based approach to execute code directly on the target architecture.
- Used to obtain dynamic information such as branching probability, typical loop counts, instruction frequencies, etc.
- Achieved typically by instrumenting the executable codes.
  - e.g., by inserting a counter at each conditional branch.
- Accuracy and performance of profiling depend on the ability to characterize the typical inputs.
- Profiling can not be used for worst case analysis.
Dynamic Analysis
Techniques that use results of information collected by analyzing the programs without executing them.

- No assumption about input data is made.
- Restriction on software
  - bounded loops
  - absence of recursive functions
  - absence of dynamic function calls

- Can be used for:
  - program analysis — behavior of a single program on a processor.
  - system performance analysis — behavior of multiple processes on a single processor or several processors.

Program Analysis

- The estimated worst-case execution time (WCET) must be safe and tight.

- The ideal tool for source code analysis would produce a good WCET estimate based on the following inputs:
  - Source code.
  - Compiler.
  - Machine architecture description.
  - Operating system.
Program Path Analysis

- To determine what sequence of instructions will be executed in the worst case scenario.

A basic block is composed of instructions in a straight line.

- Let us first assume that each instruction takes a fixed time to execute.

Program Path Analysis

- Infeasible paths can be eliminated by data flow analysis and path information provided by the programmer.

- The number of feasible paths is typically exponential with the program size.

- Efficient methods are needed to avoid enumeration of all paths.
ILP Formulation

Let $x_i$ be the number of times a basic block $B_i$ is executed; $c_i$ be the execution time of the basic block $B_i$, which is assumed to be a constant.

The total execution time of the program for a particular execution is:

$$\sum_{i=1}^{N} c_i \cdot x_i$$

$$C_1 + C_2 + C_4 + 11 \cdot C_5 + 10 \cdot C_6 + C_7$$

The estimated WCET of the program is:

subject to a set of constraints $Ax \leq b$.

- The quality of the constraints define the tightness of the estimate.
- Constraint classification:
  - Program structural constraints — deduced from the program’s control flow graph.
  - Program functionality constraints — provided by the user to specify loop bounds and other path information.
### An Example

```c
/* k >= 0 */
s = k;
while (k < 10) {
    if (ok)
        j++;
    else {
        j = 0;
        ok = true;
    }
    k++;
}
r = j;
```

### Constraints I

- **Structural constraints:**
  
  \[ \begin{align*}
  d_1 &= 1 \\
  x_1 &= d_1 = d_2 \\
  x_2 &= d_2 + d_8 = d_3 + d_9 \\
  x_3 &= d_3 = d_4 + d_5 \\
  \ldots
  \end{align*} \]
Constraints II

- Functionality constraints:
  - Loop bound information:
    \[ 0 \leq x_1 \leq x_3 \leq 10 \]
  - Path information:
    \[ x_5 \leq 1 \times x_1 \]

- Now an ILP solver can be used to find the values of \( x \)'s that lead to the maximal execution time.

- The question is how tight the result will be?

```c
/* k >= 0 */
X_s = k;
X_2 while (k < 10) {
  X_3 if (ok)
  X_4 j++;
  else {
    X_5 j = 0;
    X_6 ok = true;
  }
  X_7 k++;}
X_8 r = j;
```

Outline

- Static analysis techniques
- Worst-case execution time analysis
  - Micro-architecture modeling and analysis
- Co-simulation approaches
Dependence on System Resources

- Very large variations in program execution time can result from different uses of system resources:
  - memory references
  - pipeline utilization
- Instruction execution times are not longer constant and independent of each other.
- A brutal worst-case assumption, such as always cache miss, is too pessimistic to be practical (e.g., can be 30 times worse).
- What are needed:
  - a detailed micro-architectural model.
  - adjacent instructions should be analyzed together for pipeline performance.
  - cache access must be analyzed globally.

Cache Analysis

- Cache activity of a cache line is affected by all instructions mapping to that cache line.

Ex. A direct mapped cache:

```
Memory address = 0x020-0029
```

```
<table>
<thead>
<tr>
<th>Tag</th>
<th>Slot</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
9990-9999
9012-0129
9011-0119
9010-0109
9002-0029
9001-0019
9000-0009
```

10,000-Word Memory

```
<table>
<thead>
<tr>
<th>Tag</th>
<th>Slot No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>90-99</td>
</tr>
<tr>
<td>8</td>
<td>80-89</td>
</tr>
<tr>
<td>7</td>
<td>70-79</td>
</tr>
<tr>
<td>6</td>
<td>66-69</td>
</tr>
<tr>
<td>5</td>
<td>50-59</td>
</tr>
<tr>
<td>4</td>
<td>40-49</td>
</tr>
<tr>
<td>3</td>
<td>30-39</td>
</tr>
<tr>
<td>2</td>
<td>20-29</td>
</tr>
<tr>
<td>1</td>
<td>10-19</td>
</tr>
<tr>
<td>0</td>
<td>00-09</td>
</tr>
</tbody>
</table>
```

100-Word Cache
Cache Analysis

- Global analysis is required.
- Must be analyzed with path analysis together.
- Direct-mapped Cache Analysis:
  - Determine each instruction’s cache hit and cache miss counts.
  - Instruction in a basic block may not have the same counts.

L-block — A line block is a continuous sequence of instructions that are in the same basic block and mapped to the same cache line.

The Modified Cost Function

\[
WCET = \max \left( \sum_{i=1}^{N} \sum_{j=1}^{n_i} \left( c_{HIT_{i,j}} \cdot x_{HIT_{i,j}} + c_{MISS_{i,j}} \cdot x_{MISS_{i,j}} \right) \right)
\]

where

- \( c_{HIT_{i,j}} \) — execution time of L-block \( B_{i,j} \) with cache hit
- \( x_{HIT_{i,j}} \) — cache hit count of L-block \( B_{i,j} \)
- \( c_{MISS_{i,j}} \) — execution time of L-block \( B_{i,j} \) with cache miss
- \( x_{MISS_{i,j}} \) — cache miss count of L-block \( B_{i,j} \)
- \( x_i = x_{i,j} = x_{HIT_{i,j}} + x_{MISS_{i,j}} \) \( \{j = 1, 2, \ldots, n_i\} \)

Subject to the structural/functionality constraints, discussed before, as well as additional cache constraints.
Cache Constraints

For each cache line, three assignments can occur:

- There is only one L-block $B_{i,j}$ mapping to it — there will be at most 1 cache miss:
  \[ x_{MISS_{i,j}} \leq 1. \]

- There are two or more non-conflicting L-blocks mapping to the same cache line — when a miss occurs in either block, the L-blocks will be loaded and no more misses will occur:
  \[ x_{MISS_{i,1}} + x_{MISS_{i,2}} \leq 1. \]

- There are two or more conflict L-blocks mapping to the same cache line — the order they are executed will affect the cache hits/misses.

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>Basic Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$B_1, B_{1,7}$, $B_3, B_{3,7}$</td>
</tr>
<tr>
<td>1</td>
<td>$B_1, B_{1,2}$, $B_3, B_{3,2}$</td>
</tr>
<tr>
<td>2</td>
<td>$B_1, B_{1,3}$, $B_3, B_{3,1}$</td>
</tr>
<tr>
<td>3</td>
<td>$B_1, B_{1,2}$</td>
</tr>
</tbody>
</table>

Cache Conflict Graphs

- A CCG is constructed for each cache line containing two or more conflict L-blocks.

Possible program flow between Blocks, acquired from CFG:

\[ p(k, l, k) \]
\[ p(k, l, m) \]
\[ p(s, k) \]
\[ p(s, l) \]
\[ p(s, m) \]
\[ p(s, n) \]
\[ p(m, n, m) \]
\[ p(m, n, l) \]
\[ p(m, n, c) \]
\[ p(s, c) \]

\[ p(node1, node2) \] denotes the execution counter associated with each edge.
Cache Conflict Graphs (Cont’d)

- $s$ and $e$ nodes represents the start and the end of the program respectively.
- A node $B_{i,j}$ for each conflicting l-block.
- Edges represent possible program flow between blocks — acquired from program CFG.
- $p$ (node1, node2) is the execution counter associated with each edge.

Cache Conflict Graph Example

\[
x_i = \sum_{u,v} p(u.v, i.j) = \sum_{u,v} p(i.j, u.v)
\]
Constraints on CCG

- The execution counters are bound to the structural and functional constraints:
  - the execution count of a L-block must be equal to the execution count of the basic block;
  - the control flow to an L-block node must be equal to the flow from the L-block node

\[
x_i = \sum_{u,v} p(u,v,i,j) = \sum_{u,v} p(i,j,u,v)
\]

Advantage of the Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Measured WCET</th>
<th>Estimated WCET with cache analysis</th>
<th>Estimated WCET without cache analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check_data</td>
<td>4.41 x 10²</td>
<td>4.91 x 10²</td>
<td>11.9 x 10²</td>
</tr>
<tr>
<td>Piksrt</td>
<td>1.79 x 10³</td>
<td>1.82 x 10³</td>
<td>5.01 x 10³</td>
</tr>
<tr>
<td>Line</td>
<td>4.85 x 10³</td>
<td>6.09 x 10³</td>
<td>9.15 x 10³</td>
</tr>
<tr>
<td>Circle</td>
<td>1.45 x 10⁴</td>
<td>1.53 x 10⁴</td>
<td>1.59 x 10⁴</td>
</tr>
<tr>
<td>FFT</td>
<td>2.05 x 10⁶</td>
<td>2.71 x 10⁶</td>
<td>4.04 x 10⁶</td>
</tr>
<tr>
<td>Des</td>
<td>2.42 x 10⁵</td>
<td>3.66 x 10⁵</td>
<td>6.69 x 10⁵</td>
</tr>
<tr>
<td>Fullsearch</td>
<td>6.25 x 10⁴</td>
<td>9.57 x 10⁴</td>
<td>29.0 x 10⁴</td>
</tr>
<tr>
<td>Whetstone</td>
<td>6.83 x 10⁶</td>
<td>10.2 x 10⁶</td>
<td>14.9 x 10⁶</td>
</tr>
<tr>
<td>Dhry</td>
<td>5.52 x 10⁵</td>
<td>7.53 x 10⁵</td>
<td>13.3 x 10⁵</td>
</tr>
<tr>
<td>Matgen</td>
<td>9.28 x 10³</td>
<td>10.9 x 10³</td>
<td>17.2 x 10³</td>
</tr>
</tbody>
</table>
WCET Analysis with General Cache

- There are several variables which influence the complexity of cache analysis:
  - number of competing line-blocks \( m \);
  - cache associativity level \( n \);
  - cache replacement method.
- For LRU (least recently used), the complexity grows as:
  \[
  \sum_{i=0}^{n} \frac{m!}{(m - i)!}
  \]
- By using a more detailed level of cache modeling better estimations can be acquired, but the problem become intractable if the programs are very large.

Remarks on Performance Analysis

- One of the main issues of embedded system design is estimation and analysis.
- Analysis of average and probabilistic performance can be done by simulation.
- Worst case execution time analysis can only be efficiently done by static analysis techniques.
- Efficient techniques for analyzing impacts of many advanced micro-architecture components are still research issues.
Outlines:
- Static analysis techniques
- Worst-case execution time analysis
- Micro-architecture modeling and analysis
- Co-simulation approaches

Co-Simulation

- How the hardware and software components are simulated at the same time?

Problems:
- Different simulation platforms are used;
- Software runs fast while hardware simulation is relatively slow.
  - How to run the system simulation as fast as possible and keep the two domains synchronized?
- Slow models provide full details and produce accurate results; fast models don’t produce enough timing information and simulation is less accurate.
Approaches to Co-Simulation 1

- Gate-level model of the processor
  - Gate level simulation of the processor is very slow (tens of clock cycles/sec).
    Ex. 10 cycles/sec, 1 GHz processor ⇒ 100 million seconds (3.2 years) are needed to simulate one second of real time.
  - This provides a very accurate solution and is very simple from the co-simulation point of view.

Approaches to Co-Simulation 2

- Instruction-set architecture models
  - There is no hardware model of the target processor; the software is executed on an ISA model (usually in C); execution on the ISA model provides interface information (including timing) needed for co-simulation.
  - This is fast but timing accuracy depends on the interface information.
Approaches to Co-Simulation 3

- **Translation-based models**

  - Program running directly on host
  - Software compiled into native code for the host
  - ASIC model (VHDL)
  - VHDL simulation

  - There is no hardware model of the target processor; the software is compiled into native code for the host processor; software execution provides interface information (including timing) needed for co-simulation.

Approaches to Co-Simulation 4

- **Hardware in the loop:**

  - Combine hardware and software in one solution, by using the physical device to model its own behavior.
  - No necessarily the most accurate model!
  - An adaptor formats inputs to the physical device, applies the input, returns the resulting outputs with timing information to the simulator.
  - This is a good choice for modeling complex standard components such as microprocessors.
Approaches to Co-Simulation 5

- Mixed level simulation — to combine the strength of simulation at different levels of abstraction and provide a possibility to compare results at different levels.

- Broadband simulator — One broadband language is used which covered several abstraction levels.

- Multi-simulator — several simulators are used in an integrated environment. Main issues to deal with:
  - The data exchange between the various simulators.
  - The synchronization of the simulators, using time stamps. It allows them to proceed independently. If a signal is received with a time stamp lower than the current clock in a simulator, the simulator will have to role back.

Concluding Remarks

- Efficient techniques for static analysis and simulations are essential for guiding the design process.

- Static analysis techniques are good for extreme-case performance analysis, but work only for simple systems.

- In practice, simulations are mainly used in the industry.

- The basic problem of co-simulation is how to simulate HW and SW together so that it is fast and accurate.

- Formal verification proves design correctness.
  - Computational complexity.
  - Integration into the design flow.