Pipeline Timing Analysis Using a Trace-Driven Simulator

Jakob Engblom
jakob@docs.uu.se
Dept. of Computer Systems
Uppsala University
& IAR Systems AB
Sweden

Andreas Ermedahl
abbe@docs.uu.se
Dept. of Computer Systems
Uppsala University

What is WCET?

- WCET: Longest execution time
- Measurement unsafe, static analysis instead
- Estimate should be safe and tight
- No interrupts or context switches

Uses of WCET Information

- Scheduling
- Schedulability analysis
- Formal verification
- System dimensioning
- Modelling
- Simulation
- Program performance tuning

Phases of WCET Analysis

- Flow analysis
  - Determine the dynamic behaviour of program
- Low level analysis
  - Determine execution time for small program parts
- Calculation
  - Combining the flow- and low level analysis

Our Approach to WCET

- WCET analysis divided into modules
- Loosely coupled components
- We do not want to use a special-purpose pipeline model

Our Approach to WCET

- Focus of this paper is the pipeline analysis and its use of the simulator
- Assume cache analysis performed before the pipeline analysis
### Input to Pipeline Analysis

- **Cache analysis results:**
  - Use in pipeline analysis
- **Flow analysis results:**
  - Pass through

### Execution fact:
- Information about how an instruction executes
  - Instruction cache hit/miss
  - Memory area accessed
  - Branch taken
  - etc.

### Result of Pipeline Analysis

- **Graph expressed by constraints**
- **Flow and timing**
  - Cache information consumed
  - Flow information propagated

- **Execution counts (x)**
  - Unknown, constrained
- **Nodes and edges**
  - Isolated execution time
- **Timing effects (t)**
  - For nodes
  - Determined by PA
- **Negative overlap**

### Calculating WCET

- **WCET Formula:**
  - $WCET = \max \sum_{i=1}^{\text{edges}} (f_i + t_i)$, all entity
  - Meeting all constraints

- Solve using CSP or ILP
- Known as “IPET Method”
Simple Pipeline Overlap

- Between two execution scenarios
  - Illustrated using reservation tables for a fictive simple pipelined CPU with floats
  - (This is previous work by our group)

  ![Reservation Table](image)

Timing Effects by Simulator

- Determine timing effect by running sequences of execution scenarios
  - First individual scenarios
  - Then the whole sequence

  ![Simulator Diagram](image)

Complex Pipeline Overlap

- Problem: How do we model one execution scenario overlapping more than one successor?

  ![Complex Pipeline Diagram](image)

  Answer: Introduce timing effects (\( \delta \)) over more than just pairs of execution scenarios.

First we do the normal analysis for each pair of nodes in the sequence

- \( \delta_{AB} = t_{AB} \cdot t_A \cdot t_B \)

- \( \delta_{BC} = t_{BC} \cdot t_B \cdot t_C \)
Complex Pipeline Overlap

- Then we add a timing effect $\delta_{ABC}$
  - Represent effect of A over B into C
  - Calculate from times for subsequences

\[
\delta_{ABC} = t_{ABC} - t_{AB} - t_{BC} + t_B
\]

Notes on the Model

- $\delta_{ABC}$ may be positive
- Can handle arbitrary long sequences
- Must provide constraints for $x_{sec}$
  - Bonus: tells us when we need flow analysis results (execution constraints) for paths
- It’s all in the paper!

Power of the Model

- Handles single-issue, in-order pipelines
  - Covers most embedded pipelined CPUs in use today (ARM, SH, AVR, ...)
- May be extended to multiple-issue pipelines, but this is future work
- Out-of-order processors are also future work

Summary

- Given introduction to ideas in paper
- Contributions:
  - Using simulator instead of special-purpose pipeline model
  - Modeling pipeline effects across multiple basic blocks
  - Incorporating cache analysis results into the pipeline analysis
  - When are path constraints useful?