A Time-Predictable Execution Mode for Superscalar Pipelines

Christine Rochange and Pascal Sainrat

Traces is a Research group on Architecture and Compilation for Embedded Systems

Toulouse France
Hard Real-Time Systems

- **Worst-Case Execution Time (WCET)**
  - upper bound on the execution time, whatever the input data are

- **WCET has to be estimated for**:
  - task scheduling
  - hardware sizing

- **WCET estimation should be both safe and tight**
Outline

- WCET estimation
- Long timing effects
- Making the pipeline time-predictable
- Performance results
- Conclusions and future work
WCET estimation (1)

- **Hypotheses:**
  - **Which kind of code can we analyse?**
    - target architecture has to be known
    - « isolated » program:
      - no context switching
      - no interrupts
  - **Inter-task interferences**
    - context-switch cost
      - out of the scope of WCET estimation (handled by the scheduler)
    - functional interferences
      - might generate errors in WCET estimation (e.g. caches)
      - should be controlled by the programmer (e.g. cache locking)
Solution 1: measuring all the possible paths

- requires data sets that ensure full path-coverage
  - can we generate these data sets?
  - solution: symbolic execution paradigm
    - input data are unknown
    - unknown values are propagated through the computation
    - whenever a branch is executed with an unknown condition, explore both branches

- is costly in time
  - the number of paths is generally high
WCET estimation (3)

- **Solution 2: static analysis**
  - **flow analysis**
    - based on the CFG and/or the syntax tree
    - computes loop bounds, infeasible paths, ...
  - **low-level analysis**
    - evaluates the execution time of code units (basic blocks)
      - first step: caches, branch predictor, ...
      - second step: pipeline
  - **WCET computation**
    - combines the results of the preliminary analyses
      - based on the syntax tree (Extended Timing Schema)
      - … or on the CFG (Implicit Path Enumeration Technique)
**IPET (1)**

**Implicit Path Enumeration Technique**

- **Execution time:**
  
  \[ T = \sum x_i \cdot t_i + \sum x_{i,j} \cdot t_{i,j} \]

  \( x_i = \text{number of executions} \)
  
  \( t_i = \text{execution time} \)

- **To compute the WCET:**
  - find the \( x_i \)'s and the \( x_{i,j} \)'s that maximize the execution time
  - the \( x_i \)'s and the \( x_{i,j} \)'s are linked
  - some \( x_i \)'s and \( x_{i,j} \)'s must be upper-bounded
Objective function:
\[ \max T = \sum x_i t_i + \sum x_{i,j} t_{i,j} \]

Structural constraints:
\[ x_0 = 1 \]
\[ x_0 = x_{0,1} \]
\[ x_1 = x_{0,1} \]
\[ x_1 = x_{1,2} + x_{2,3} \]
\[ x_2 = x_{1,2} \]
\[ x_2 = x_{2,6} \]
\[ x_3 = x_{1,3} \]
\[ x_3 = x_{3,4} + x_{4,5} \]
\[ x_4 = x_{3,4} \]
\[ x_4 = x_{4,5} \]
\[ x_5 = x_{4,5} + x_{3,5} \]
\[ x_5 = x_{5,1} \]
\[ x_6 = x_{2,6} \]
\[ x_6 = 1 \]

Flow constraints:
\[ x_{5,1} < 8 \]

ILP solver
e.g. lp_solve
Outline

- WCET estimation
- Long timing effects
- Making the pipeline time-predictable
- Performance results
- Conclusions and future work
Long Timing Effects (1)

\[ \delta_{ABC} = +2 \]

\[ t_A = 6 \]
\[ t_B = 6 \]
\[ t_C = 6 \]

\[ \delta_{AB} = -5 \]
\[ \delta_{BC} = -5 \]

Long timing effect: \[ t_{ABC} = 6 - 5 + 6 - 5 + 6 = x \]
Long Timing Effects (2)

- Engblom’s model [Engblom, 2002]

\[
t_{1\ldots n} = \sum_{i=1}^{n} t_i + \sum_{1 \leq j \leq k \leq n} \delta_{j\ldots k}
\]
Sources of LTE’s

- Structural hazards
  - superscalar pipeline
    - size of blocks compared to pipeline width
  - long-latency functional units
  - queues (e.g. fetch queue, reorder buffer, etc.)
  - specific limitations (e.g. number of pending branches)
  - ...

- Data hazards

- Ordering constraints
  - e.g. in-order completion
Are LTE’s frequent?

- **Methodology**
  - processor:
    - 6-stage 4-way OOO pipeline, perfect caches, gshare
  - benchmarks:
    - crc, jfdctint, ludcmp, fibcall, matmul, fft1, lms
  - simulator:
    - cycle-accurate, extracts sequences from the CFG

- **Results**

<table>
<thead>
<tr>
<th># LTE’s</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+4</td>
<td>+5</td>
<td>+6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>280</td>
<td>55</td>
<td>25</td>
<td>16</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># LTE’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
Can we include LTE’s in the WCET estimation?

- LTE’ are difficult to:
  - model
    - (constraints)
  - measure
    - requires simulating every possible sequence (of any length)

- Solution:
  eliminate them!

instruction flow regulation
Outline

- WCET estimation
- Long timing effects
- Making the pipeline time-predictable
- Performance results
- Conclusions and future work
Instruction-flow Regulation

- **Basic principle:**
  - **fetch gating** = delay the fetch of a new BB
  - until when?
    - no LTE can arise
    - i.e. pending blocks cannot « distort » any new block
Regulation policy

- The gating decision is based on:
  - the earliest resource requirements of any instruction that might enter the pipeline at the next cycle
    - pipeline stages, functional units, register contents, ...
  - the latest availability dates of these resources

Fetching a new basic block is delayed until they match
Implementing the regulation policy (1)

- How can the hardware know the availability dates?
  - with regular dynamic scheduling
    - has to wait that all of the instructions have been issued to the FUs
    - would seriously degrade performance
  - with dynamic prescheduling [Canal&Gonzalez,2001]
    - instructions are prescheduled when inserted in the ROB
      → prescheduling buffer
    - not optimal but:
      - simplifies the issue logic (wake-up + select)
      - allows the estimation of the availability dates in advance
Implementing the regulation policy (2)

- **Availability tables**
  - for physical registers
  - for pipeline stages
  - for functional units
  - updated as the instructions are prescheduled

- **Identification of (compiler-defined) basic blocks**
  - the compiler should mark the end of basic blocks
    - with a systematic branch
  - at the decode stage → the gate is placed at this stage
Outline

- WCET estimation
- Long timing effects
- Making the pipeline time-predictable
- Performance results
- Conclusions and future work
Cost in performance (1)

- Degradation due to dynamic prescheduling
  - from 5% (2-way superscalar) to 10% (8-way)

- Degradation due to instruction-flow regulation (including dynamic prescheduling)

Mean total loss:

- 19.1% (2-way)
- 37.6% (8-way)
Cost in performance (2)

- Cost should be put into perspective
  - to get safe WCET estimations, the use of simple processors is usually recommended
    - some simple pipelines (scalar, in-order execution) can be proved « LTE-free »

- a regulated dynamically-prescheduled pipeline gives better performance
Outline

- WCET estimation
- Long timing effects
- Making the pipeline time-predictable
- Performance results
- Conclusion
Instruction-flow regulation in brief

- Eliminates long timing effects
- Has a cost in performance …
- … but makes it possible to get some performance
- Could easily be disabled
  → two execution modes
    - high-performance
    - real-time
Related work

- **AbsInt**
  - overview:
    - uses Abstract Interpretation to determine the possible states of the pipeline at each point of the code
    - computes the maximum execution time of each basic block
    - uses IPET to compile individual execution times
  - analysis would be simpler with a regulated pipeline

- **VISA**
  - overview:
    - WCET is computed assuming a simple predictable pipeline
    - at runtime, the execution time (on a high-performance pipeline) is compared to the estimated WCET at some checkpoints in the code
    - in case of overrun, the pipeline switches to the predictable execution mode
  - the regulated execution mode could be the predictable mode
Conclusion and future work

- High-performance processors used in hard real-time systems
- Timing predictability is one of the major issues
- Our research activities include:
  - proposing new processor architectures that fit predictability requirements
  - improving current models to take into account high-performance schemes in WCET analysis