A Flexible Tradeoff between Code Size and WCET Using a Dual Instruction Set Processor

Sheayun Lee, Jaejin Lee, Chang Yun Park, and Sang Lyul Min
Seoul National University & Chungang University
September 3, 2004
SCOPES 2004, Amsterdam, The Netherlands
Motivation

Embedded systems are often constrained in terms of both:

- code size
  due to a limited amount of available memory
- execution time
  due to real-time nature of applications

Both constraints should be considered at the same time, in order to build cost-effective systems.
Dual Instruction Set Processors

Full instruction set
- large instructions (usually 32 bits / instruction)
- faster execution

Reduced instruction set
- small instructions (usually 16 bits / instruction)
- slower execution in exchange for smaller code size

Provides a mechanism for tradeoff between code size and execution
Prior Work

Selective code transformation

• code generation for dual instruction set processors
  aimed at improving the average case performance

• detailed cost-benefit model based on profile information

• reference:

  S. Lee, J Lee, S. L. Min, J Hiser, and J W. Davidson. Code Generation for a Dual Instruction Processor Based on Selective Code Transformation. [SCOPES 2003]
Average Case Optimization vs. Worst Case Optimization

Average case performance optimization

- traditional compiler optimizations
- can be based on profile information

Worst case performance optimization

- targeted towards enhancing WCET (worst case execution time)
- should be based on WCEP (worst case execution path) information
Our Goals

Provide a mechanism to enable a flexible tradeoff between code size and WCET
Enable optimization towards enhancing the worst case performance of programs

→ A framework that can be used to fine-tune an application program on a spectrum of code size and worst case performance.
Selective Code Transformation

- Selective Code
  - Transformation
    - full ISA code
    - code size diff
    - exec time diff
    - constraint on total code size
    - source program
      - compiled into reduced ISA code
    - transform all blocks
      - WCET analysis
      - annotated syntax tree: WCEP and frequency information
      - selection algorithm for blocks to be transformed: path-based profitability analysis
      - timing update
      - code transformer
        - mixed mode code
        - selective code transformation
Path-Based Profitability Analysis

In determining which blocks are to be transformed, basic blocks should not be considered individually

- insertion of mode switch instructions
  - code size overhead
  - execution time overhead
- profitability (reduction of WCET for code size increase) assessed for each acyclic subpath of the WCEP
Cost-Benefit Model

\[ c(p) = \sum_{v \in V(p) \cap R} (s_F(v) - s_R(v)) + o_s \times \left( \left| E^M(p) \right| - \left| E^m(p) \right| \right) \]

- **Cost of transforming blocks on path** \( p \)
- \( s_F(v) - s_R(v) \): sum of code size differences for blocks being transformed
- \( o_s \): mode switching overhead in terms of code size

\[ b(p) = \sum_{v \in V(p) \cap R} (c_v(v) \times (t_R(v) - t_F(v))) - o_t \times \left( \sum_{e \in E^M(p)} c_E(e) - \sum_{e \in E^m(p)} c_E(e) \right) \]

- **Benefit from transforming blocks on path** \( p \)
- \( c_v(v) \times (t_R(v) - t_F(v)) \): reduction of WCET by transforming the blocks
- \( o_t \): mode switching overhead in terms of execution time
Selection Algorithm: Greedy Heuristic

\[ B \leftarrow U_s - S_R \]
\[ R \leftarrow V \]
\[ F \leftarrow \emptyset \]
\[ P \leftarrow \{ \text{acyclic subpaths of the WCEP} \} \]

while (there exists a path \( p \in P \) s. t. \( c(p) \leq B \) and \( b(p) \geq 0 \)) {

for each \( p \in P \), calculate \( r(p) = b(p) / c(p) \)

select \( p \in P \) with maximum \( r(p) \) with \( c(p) \leq B \)

\[ B \leftarrow B - c(p) \]
\[ F \leftarrow F \cup V(p) \]
\[ R \leftarrow R - V(p) \]

if (change in WCEP)

\[ P \leftarrow \{ \text{acyclic subpaths of the new WCEP} \} \]
else

\[ P \leftarrow P - \{ p \mid V(p) \cap R = \emptyset \} \]
Information Update

In each iteration of the selection algorithm, the following must be updated:

- Cost and benefit associated with each subpath
  since transforming a subpath may affect the cost and benefit of other subpaths
  → re-calculate them in each iteration

- WCEP information
  since the previous WCEP is no longer guaranteed to have the largest execution time, because the WCET has been reduced
  → timing update using an annotated syntax
Hierarchical WCET Analysis

A set of timing formulas for different types of program constructs, operating on each node in the syntax tree.

Extensions to account for history-sensitive timing variations

- PA (path abstraction) and WCTA (worst case timing abstraction)
- concatenation and pruning operations → effectively enumerate all the possible execution paths in the program
Timing Update

Timing update information is propagated upwards in the syntax tree

- begin from the leaf nodes (basic blocks) whose execution times have been changed
- WCTAs for nodes encountered are updated until the root node is reached

Need a data structure to minimize the re-calculation of WCTAs for nodes associated with tree.
Annotated Syntax Tree

WCTA associated with each node is recorded (annotated) on the syntax tree

- WCTAs are reused for nodes that are not affected by the transformation

→ Re-analyze only the intermediate nodes on the path from the root node to leaf nodes corresponding to the blocks being transformed
Timing Update – Example

control flow graph

annotated syntax tree
Nodes are processed in an order of non-increasing depth
• so that no node is visited more than once
The number of intermediate nodes whose WCTAs are re-evaluated over all the iteration of the selection algorithm is bound by $O(n^2)$
Implementation

Targeted to ARM7TDMI processor
Path-based profitability analysis and the greedy selection algorithm
Hierarchical WCET analysis and timing update algorithm
Code generator based on VPO
  • instruction selection mechanism based on peephole optimization
Experiments

Benchmark programs

- extracted from SNU-RT real-time benchmark suite – fir, matmul, ludcmp, jfdctint

11 different versions for each program

- T: compiled entirely into Thumb instructions
- A: all blocks transformed into ARM instructions
- M1 ~ M9: generated by the proposed
Experiments (Cont’d)

For each of the different version of code, we compared

- relative code size

\[
\text{relative\_size} (M) = \frac{\text{size} (M) - \text{size} (T)}{\text{size} (A) - \text{size} (T)}
\]

- relative WCET

\[
\text{relative\_WCET} (M) = \frac{\text{WCET} (M) - \text{WCET} (A)}{\text{WCET} (T) - \text{WCET} (A)}
\]
Results (1/2)

(a) fir

(b) matmul
Results (2/2)

(c) ludcmp

(d) jfdctint
Conclusions

Flexible tradeoff between code size and WCET using a dual instruction set processor
- path-based profitability analysis
- greedy selection algorithm

Program optimization towards enhancing the WCET
- guided by timing update that captures the changes in WCET and WCEP
- using hierarchical WCET analysis and an annotated syntax tree
Future Work

Incorporation of instruction caches
  • analysis in conjunction with a code placement technique
Enabling program optimization techniques that operate across basic block boundaries
  • e. g. global register allocation
System-level optimization for tradeoff among size, time, and energy
  • involving multiple applications