Code Generation for a Dual Instruction Set Processor Based on Selective Code Transformation

Sheayun Lee, Jaejin Lee, Sang Lyul Min,
Jason Hiser, and Jack W. Davidson
Seoul National University & University of Virginia

Sep. 24, 2003
SCOPES 2003, Wien, Austria

Motivation

- Code size is often a critical constraint in cost-sensitive embedded systems.
  - Due to a limited amount of available memory
- Dual instruction set processors provide an effective mechanism to trade performance off for a small code size.
  - Ex. Thumb programs are typically 30% smaller and run 25% slower than their ARM counterparts.

Dual Instruction Set Processors

- Full instruction set
  - Normal instructions, all operations available
  - Relatively rich addressing modes
  - Full set of general purpose registers available
- Reduced instruction set
  - Typically a subset of the full instruction set
  - Shorter instructions (usually half – 16 bits)
  - Limited operations and addressing modes
  - Only a subset of GP registers made visible
- Examples
  - ARM/Thumb, MIPS 32/16-TinyRISC, ARC Tangent, …

Comparison of the two ISA

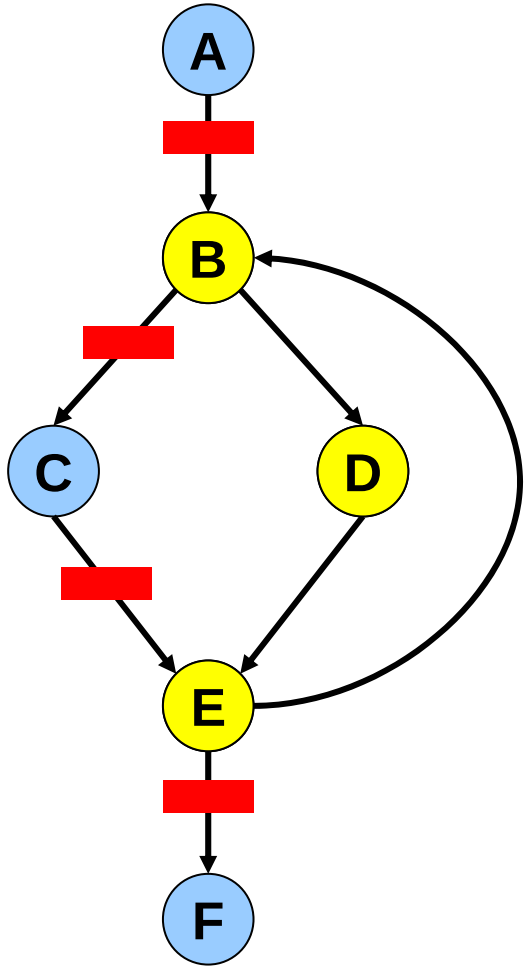
- Full instruction set
  - Larger code size, faster execution
  - Used for time-critical functions
    - Real-time data processing
    - Interrupt/exception handlers
- Reduced instruction set
  - Smaller code size, slower execution
  - Used for non-time-critical functions
    - User interface functions
    - Bookkeeping code

Coarse-Grained Approach

- Current tool support from ADS (ARM Developer Suite)
  - Instruction set mode specified at the module (file) level
    - Provided as a command-line switch to the compiler
  - Linker support to handle the dynamic mode transitions
  - Called ARM/Thumb interworking

Fine-Grained Approach

- Observation
  - Function-level or module-level approaches are too coarse-grained for enabling a flexible tradeoff between code size and execution time.
- More fine-grained approach is required.
  - Mode switches must be explicitly handled by code generation.



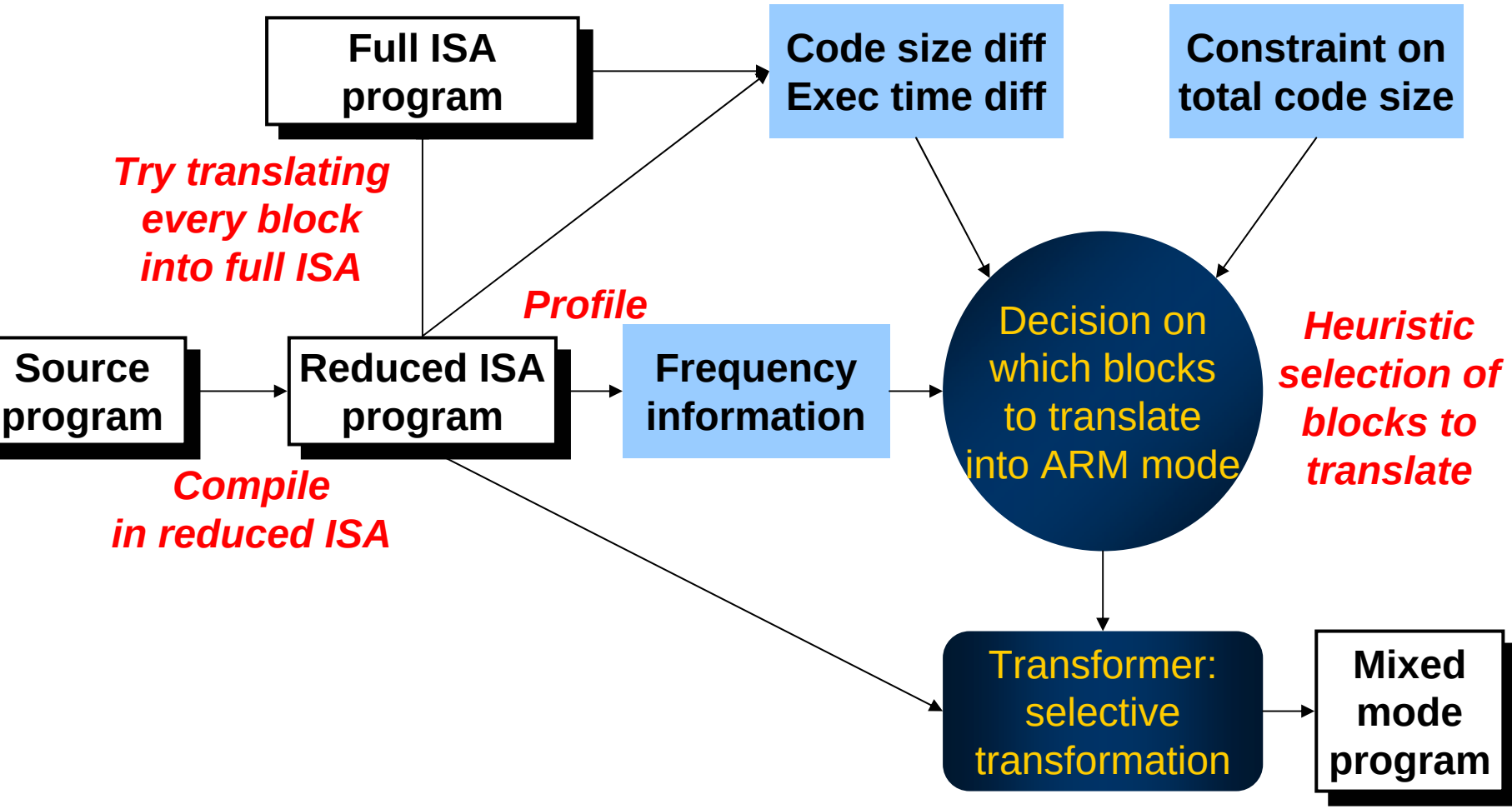
Our Objective

- Given a program, generate code that
  - maximizes performance
    - in terms of execution time
  - while satisfying code size requirement
    - given as an upper bound on the instruction space
  - by selectively using the two different instruction sets for different program parts in a single program (even inside a single function)
    - taking the mode switching overhead into account

Intuitive Solution

- Which instruction set should be used for which part of a given program?
  - Use the full instruction set for
    - time-critical portion of the code (frequently executed)
    - thereby maximizing the performance
  - Use the reduced instruction set for
    - all the other parts (infrequently executed)
    - thereby minimizing the code size

Selective Code Transformation



Program Notations

- A program is given by $P = <V, E>$, where
  - $V = \{v_i \mid i = 1, 2, \ldots, n\}$
  - $E = \{e_{ij} = <v_i, v_j> \mid$ there exists a control flow $v_i \rightarrow v_j\}$
- Code size and execution time variables for each basic block v

Instruction set	Reduced	Full
Code size	$s_R(v)$	$s_F(v)$
Execution time	$t_R(v)$	$t_F(v)$

Formal Problem Description

- Find an assignment of instruction set for each basic block that maximizes performance while satisfying code size constraint.

$$f : V \rightarrow \{\alpha, \beta\}$$

$$f(v_i) = \begin{cases} \alpha \text{ if } v_i \text{ is to be compiled into full instructions} \\ \beta \text{ if } v_i \text{ is to be compiled into reduced instructions} \end{cases}$$

- Such an assignment partitions the set of blocks into two disjoint subsets.
  - $F = \{v \mid f(v) = \alpha\}$
  - $R = \{v \mid f(v) = \beta\}$

Code Size and Execution Time

Total code size

Code size overhead due to mode switches

$$S = \sum_{v \in F} s_F(v) + \sum_{v \in R} s_R(v) + s^*$$

$$\Delta_t = \sum_{v \in F} c_V(v) \times (t_R(v) - t_F(v)) - t^*$$

Execution time savings

Execution frequency of block v

Execution time overhead due to mode switches

Mode Switching Overhead

Set of edges along which mode switches should occur

$$E^* = \left\{ e_{ij} \in E \mid (v_i \in F \wedge v_j \in R) \vee (v_i \in R \wedge v_j \in F) \right\}$$

Execution frequency of edge e

Total code size overhead →

$$s^* = o_s \times \left| E^* \right|$$

Total execution time overhead →

$$t^* = o_t \times \sum_{e \in E^*} c_E(e)$$

Optimization Problem

Given a program $P = <V, E>$, find an assignment $f : V \rightarrow \{\alpha, \beta\}$ such that it maximizes

$$\Delta_t = \sum_{v \in F} c_V(v) \times (t_R(v) - t_F(v)) - o_t \times \sum_{e \in E^*} c_E(e)$$

while satisfying

$$S = \sum_{v \in F} s_F(v) + \sum_{v \in R} s_R(v) + o_s \times \left| E^* \right| \leq U_s$$

Upper bound on the maximum code size for the whole program

Approximation Method

- Path-based approach to selective code transformation
  - Based on intraprocedural acyclic subpaths
    - Acyclic subpaths capture the set of basic blocks that are executed together.
  - Enumerate the acyclic subpaths and associate with each of them
    - Cost (increase of code size)
    - Benefit (reduction in execution time)
  - A greedy heuristic incrementally selects one path at a time, whose blocks are to be transformed.

Path-Based 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

Set of edges where mode switch instructions are newly introduced

Set of edges where previously existing mode switch instructions are removed

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

Benefit from transforming blocks on path p

Set of blocks on path p that are currently in the reduced instruction set mode

Selection Algorithm: Greedy Heuristic

$B \leftarrow U_s - S_R$
$R \leftarrow V$
$F \leftarrow \varnothing$
$P \leftarrow$ {intraprocedural acyclic subpaths}

do {
    for each $p \in P$ calculate $r(p) = b(p) / c(p)$
    select $p \in P$ with maximum $r(p)$ with $c(p) \leq B$
    $B \leftarrow B - c(p)$
    $F \leftarrow F \cup V(p)$
    $R \leftarrow R - V(p)$
    $P \leftarrow P - \{ p \mid V(p) \cap R = \varnothing \}$
} while ($B \geq \min_{p \in P}\{ c(p) \} \wedge \min_{p \in P}\{ c(p) \} \geq 0 \wedge R \neq \varnothing$)

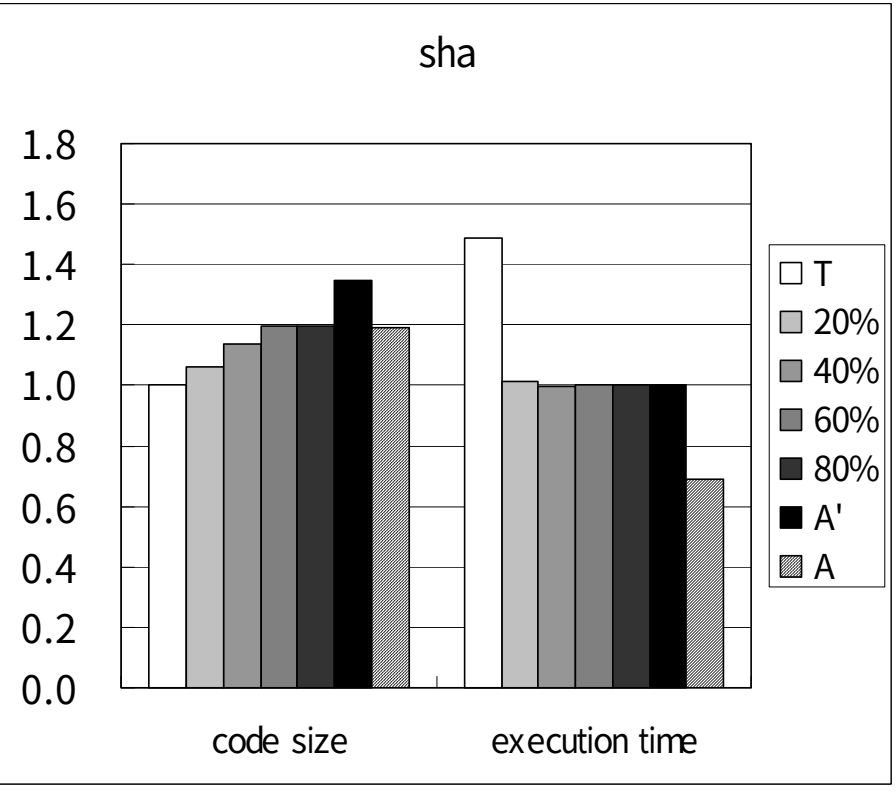
Implementation

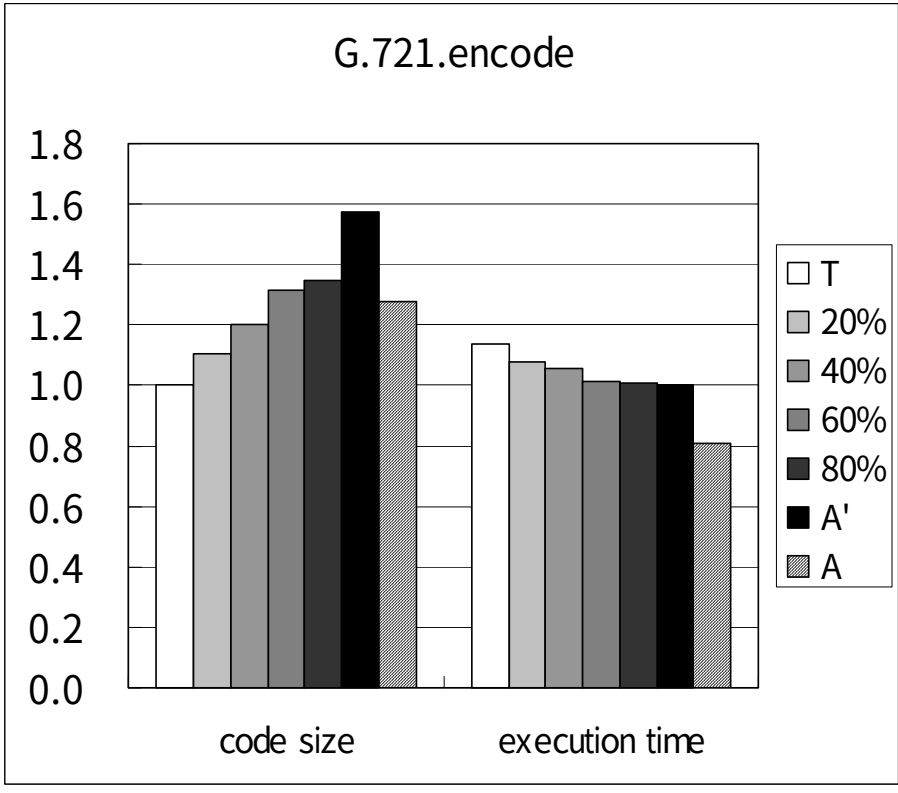
- Based on vpo (very portable optimizer), targeted for ARM/Thumb architecture
  - RTL representation of programs
  - Instruction selection mechanism based on peephole optimization
    - One or more Thumb RTL statement can be combined to form a single ARM RTL statement.
  - Automatic insertion of mode switching instructions
    - Based on control-flow analysis

Experiments

- Test programs are taken from
  - MiBench
  - MediaBench
- Six different versions of code are generated for each of the test programs.
  - With different code size limits
- Each test program is run on an evaluation board for execution time measurement.
  - XScale core-based PXA250 processor
  - Execution times are measured by using the `gettimeofday ()` system call.

Results


sha
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
code size
execution time
T
20%
40%
60%
80%
A'
A


G.721.encode
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
code size
execution time
T
20%
40%
60%
80%
A'
A

Conclusions

- Code generation framework that can exploit dual instruction sets
  - Path-based cost-benefit model
  - Heuristic selection of instruction set mode for each basic block
  - Automatic handling of dynamic mode switches

Future Work

- Efficient post-pass register allocation algorithm
  - Exploit all the available registers in the transformed code sections
  - Need to precisely analyze the impact of allocating each program variable to a register
  - Further enhance the performance of the resulting mixed instruction set mode program