Code Re-ordering for a Class of Reconfigurable Microprocessors

Brian F. Veale[1], John K. Antonio[1], and Monte P. Tull[2]

[1] School of Computer Science
[2] School of Electrical and Computer Engineering
University of Oklahoma, Norman, Oklahoma, USA 73019
{veale, antonio, tull}@ou.edu

A class of reconfigurable processors is introduced in which support for an instruction set is distributed among a collection of pre-defined configurations. For this new class of reconfigurable processors, there is assumed to be a pre-defined collection of configurations in which each configuration supports a subset of the overall instruction set. The union of all subsets of instructions, associated with the configurations, defines the instruction set supported by the reconfigurable processor. An objective for this class of reconfigurable processors is the support of popular commercial instruction set architectures with less hardware than required using existing static (i.e., non-reconfigurable) processors.

The basic problem considered is the following: Given a collection of configurations for a reconfigurable processor and given an executable program based on the entire instruction set, devise an algorithm to re-order the given program code so as to minimize the number of configuration switches required to execute the program on the reconfigurable processor. Our solution approach is based on a block-based analysis of the program. A greedy algorithm that works on a precedence DAG of a block of code is used to schedule the instructions to minimize the total number of configuration switches for each block.

An experimental study has been conducted based on a proposed partitioning of the PowerPC™ instruction set into two mutually exclusive subsets, one consisting of all floating-point instructions and the other consisting of all other instructions. The greedy algorithm is used to re-order machine code to minimize the number of reconfigurations required. The greedy algorithm reduces the number of configuration switches by around 50% in some cases. Additionally, the results of the experimental study highlight that a low percentage of blocks, about 20%, require configuration switching for the partitions assumed. In the study, static analysis of machine code was performed, i.e., the number of times a given block is performed during an actual program execution is not considered. The processor model is assumed to consist of one execution unit that can be reconfigured to implement different partitions of the instruction set.

Future investigations will involve the tracing of program executions so that the number of times each block is actually executed can be measured. Reduction in the number of reconfigurations for blocks that are executed multiple times provides greater improvement in overall performance than gains reported here. Another area of work includes investigating the application of clustering techniques to define near-optimal instruction partitions.