Instruction Re-encoding Facilitating Dense Embedded Code

Talal Bonny and Jörg Henkel - DATE - 2008
University of Karlsruhe, CES - Chair for Embedded Systems, Karlsruhe, Germany
{bonny, henkel} @ informatik.uni-karlsruhe.de
Resumido por: Kleber Sacilotto de Souza; RA: 024249

A redução do tamanho do código das aplicações é uma das restrições importantes no design de sistemas embarcados. Neste artigo, os autores introduzem uma nova e eficiente abordagem suportada por harware, que se baseia na recodificação de bits não usados (chamados de bits *recodificáveis*).

Sistemas embarcados geralmente usam um processador relativamente lento a pouca memória para minimizar os custos. Para diminuir o uso de memória, diversas técnicas são utilizadas para reduzir o tamanho do software embarcado realizando a compressão *offline* e a descompressão *online* do código. A idéia de usar compressão de código como uma ferramenta para a redução do tamanho do chip de memória tem levantado interesse na área de processadores que realizam o *issue* de uma única instrução por vez (geralmente os processadores RISC).

As técnicas de compressão podem ser utilizadas tanto quando o ISA (*Intruction Set Architecture*) é especificado ou não. Quando o ISA é especificado, a técnica de compressão de código utiliza as informações dos *opcodes* e do formato das instruções para desenvolver o *hardware* de descompressão. Quando o ISA não é especificado, a técnica de compressão de código pode utilizar os métodos tradicionais de compressão de dados. O decodificador neste caso será mais simples do que no caso anterior, porém o código comprimido não é tão eficiente quanto à técnica com um ISA específico.

Quando uma técnica de compressão é utilizada, são geradas as instruções comprimidas e uma tabela de decodificação. Neste artigo, é utilizada uma nova técnica de compressão de código (suportada por *hardware*) para reduzir o tamanho da tabela de decodificação. Esta técnica é baseada no formato das instruções e na própria aplicação para alcançar uma alta taxa de compressão. O ponto crucial da técnica é encontrar a posição dos bits no formato das instruções que são apropriados à recodificação. Esses bits, chamados de bits *recodificáveis*, são bits do formato das instruções que podem ser recodificados devido ao fato de não serem utilizados para a decodificação da instrução, mas somente para manter o alinhamento da palavra na memória. A recodificação destes bits não deve ter nenhum efeito na funcionalidade das instruções.

Na técnica de compressão de código apresentada são realizados os seguintes passos: (1) O formato das intruções do código original é analisado e os bits *recodificáveis* são detectados. Esses bits são então substituídos por símbolos *não-importa* 'X'. Três técnicas diferentes são aplicadas para aumentar o número de bits *recodificáveis* detectados: (1.1) a maioria das aplicações não utilizam todos os *opcodes* disponíveis, portanto os *opcodes* utilizados pela aplicação são recodificados para utilizarem o menor número possível de bits. (1.2) Os grupos de intruções são analisados e os campos de registradores de cada instrução que não são utilizados são substituídos por 'X'. (1.3) Os campos de imediato e de descolamento do endereço alvo geralmente ocupam somente os bits menos significativos dos campos, deixando os bits mais significativos não usados ou usados com menor freqüencia. Estes bits são analisados e a seqüencia de bits mais freqüente é substituída por símbolos 'X'. (2) O segunto passo consiste em aplicar o algorítmo de codificação de *Huffman* para comprimir o código de instruções gerado no passo 1. Como resultado, são geradas as intruções comprimidas e uma tabela de decodificação. As instruções comprimidas são utilizadas como índices para as intruções originais que são armazenadas na tabela de decodificação. (3) As intruções comprimidas são armazenadas de maneira não contígüa na tabela de codificação, o que ocupa espaço em memória e diminui os benefícios que podem ser alcançados com a compressão de código. Para contornar este problema é utilizado o algorítmo de codificação de *Huffman* canônico. As intruções são recodificadas te tal modo que as intruções com o mesmo tamanho são representações binárias de inteiros consecutivos. A redução do tamanho das tabelas de decodificação pode também ser alcançada reduzindo o tamanho de suas colunas. E para isso é utilizado um método de compressão desenvolvido pelos autores baseado em *Look-Up Table*, que, entre outras operações, realiza a recodificação dos bits *recodificáveis*.

A decodificação das intruções é realizada por um *hardware* que realiza a decodificação em dois estágios. No primeiro estágio, o tamanho do código comprimido é calculado. Isso pode ser feito usando um comparador para cada tamanho de instrução. O segundo estágio consiste em recuperar a instrução original da tabela de decodificação especificada. Após a decodificação da instrução, é necessário uma pequena modificação no código para que ela corresponda ao formato original da instrução.

Os resultados foram obtidos de experimentos conduzidos em duas arquiteturas de processadores embarcados: MIPS (4KC) e ARM (SA-110). Para ambas as arquiteturas foi utilizada a *suite* de *benchmark MiBench*. Dos resultados experimentais pode-se observar: (1) O tamanho dos bits *recodificáveis* nas intruções podem chegar até 26% e 22% do tamanho da instrução, para os processadores MIPS e ARM, respectivamente. (2) Recodificar os bits *recodificáveis* pode reduzir o tamanho da tabela de decodificação em uma média de 37% e 35% para os processadores MIPS e ARM, respectivamente. (3) O código comprimido contém as instruções comprimidas e a tabela de decodificação. Portanto, recodificando os bits *recodificáveis* reduz o tamanho do código comprimido pois a tabela de decodificação é reduzida, conseqüentemente melhorando a taxa de compressão. A taxa de compressão foi em média de 45% para o processador MIPS e 48% para o processador ARM, dependendo do tamanho da aplicação e do formato das intruções do processador. (4) E por fim, uma pequena perda de performance pode ser notada devido ao fato de o *hardware* de decodificação introduzir uma latência de 4 *ns* a cada vez que uma intrução de *branch* ocorre.