Cool Code Compression for Hot RISC

Advanced VLSI Computer Architecture Term Project Proposal

Mark Hampton and Michael Zhang
MIT Laboratory for Computer Science, Cambridge, MA 02139
{mhampton|rzhang}@lcs.mit.edu

1 Introduction and Background

Over the next few years, more than half of the processor market will be for cost-sensitive, low-power embedded processors. Embedded processors are highly constrained by cost, power, and size, thus code size has become a very important constraint for software designers targeting embedded applications. Even though microprocessors employing RISC instruction sets benefit from its simplicity in decoding logic, low code density has frequently limited its instruction cache bandwidth, often resulting in lowered performance. High performance systems can also benefit from smaller code size due to the reduction in instruction cache miss rate. The metric for a good code compression algorithm is measured by compression ratio, shown in Equation 1, while other parameters also need to be considered such as speed and simplicity of decoding.

$$compressionratio = \frac{compressedsize}{originalsize} \tag{1}$$

Traditionally, there are two kinds of compression methods: *statistical* and *dictionary*. Statistical compression looks at the entire program and replaces more frequently appeared text patterns with shorter codewords. A good example is the Huffman encoding algorithm. Dictionary compression, on the other hand, uses fixed length codeword to act as index into the dictionary table. It is obvious that statistical compression will be able to achieve better compression ratio, but dictionary compression will give simpler and faster decoding.

2 Related Work and Our Proposal

There has been extended research done for code compression, mostly focusing on compression ratio. One category of compression concentrates on improving the RISC instruction set such as Thumb and MIPS16. These techniques bare the entire compression task to compilers and usually obtain good compression ratio. They need to execute more instructions but achieve a higher instruction fetch bandwidth.

Another kind of compression technique involves hardware decoders which decodes compressed instruction at run-time. The compressed code RISC Processor (CCRP) is such an example. No hardware or instruction set modifications is necessary. However, this method does not improve instruction fetch bandwidth since decoding is done on the cache side.

In our project, we would like to explore new ways to encoding programs We will first investigate some of the existing compression algorithms. Based on current techniques, we will examine how compilers can generate more compression-friendly code such that the compression algorithm will find more opportunities to compress the code. We will also examine how simple, fast, and low-power run-time decoding can be done, most likely in hardware. We do not intend to modify the RISC instruction set for simplicity reasons.