Using the Entropy in the SPARC Instruction Set

Olaf S. Schoepke
School of Mathematical Sciences
University of Bath
Bath, Avon BA2 7AY, United Kingdom

Abstract

This paper analyses entropy in an instruction set used to implement software on the SPARC 1 architecture. Statistics about static and dynamic characteristics of programs are used to get information about entropy, redundancy and instruction dependence of instruction sets. Instruction streams are shown to contain on average less than one bit information when treated as high order Markov source instructions. With such low entropies we can achieve much smaller static and dynamic code, due to more compact encoding. Arithmetic coding, a coding technique which can get as close as desired to the entropy, is used to encode and decode the instruction stream. Smaller code size means smaller memory requirements, less swapping, and lower memory/CPU bus traffic.

1 Introduction

Complex Instruction Set Computers (CISCs) have a lot of different instructions and the general trend in computer architecture used to be to increase the complexity. Microcode was an implementation technique that greatly facilitated this trend. The appearance of Reduced Instruction Set Computers (RISCs) [8] encouraged another trend towards simpler instructions. This paper describes an analysis of one of these RISC instruction sets with regard to entropy [9]. It is shown that there is great redundancy in these instructions and that they have extremely low entropy if treated as high order Markov sources [1]. Such low entropy implies we can get much smaller static and dynamic code size by suitable encoding. Reduced static code size means smaller memory requirements for compiled code, reducing disk space, disk traffic and virtual memory requirements. Reduced dynamic code size means less memory/CPU bus traffic during program execution of particular importance with shared memory machines. This low entropy when instruction streams are treated as high order Markov sources indicates that there is high dependency between instructions. Therefore instructions contain more information than just their own function, since they also constrain the possible instruction that might be executed next. With arithmetic coding we are able to encode the instruction stream as close as desired to the entropy.

2 Information theory

The relationship between probabilities and codes was established in Shannon's [9] source coding theorem, which shows that a symbol expected to occur with probability \( P \) is best represented in \(-\log_2 P\) bits. Thus a symbol with a high probability is coded in few bits, whereas an unlikely symbol requires many bits. A computer program is assumed to consist of a stream of symbols (the instructions) from a given alphabet \( S \) of \( q \) symbols \( S = \{s_1, s_2, \ldots, s_q\} \). If symbol \( s_i \) occurs, we obtain an amount of information \( I(s_i) = -\log_2(P(s_i)) \) bits, where \( P(s_i) \) is the probability that this will happen. The probability of occurrence of a symbol \( s_i \) is assumed to be independent of previous symbols. An optimal encoding would use \( I(s_i) \) bits to encode symbol \( s_i \).

The average amount of information obtained per symbol is equal to \( H(S) = -\sum_{i=1}^{q} P(s_i)\log_2 P(s_i) \) bits, the entropy of the zero memory source. By zero memory we mean the probability of \( s_i \) occurring is independent of those preceding it. Lower entropy means more compact encoding is possible because the entropy gives the number of bits for optimal encoding. For example, if \( S \) takes on one of three values with probability 1/2, 1/4 and 1/4, the information conveyed by the occurrence of each of these values is 1 bit, 2 bits and 2 bits, respectively. The expected information conveyed by a value of \( S \) is 1.5 bits.

In the case of instruction sets, the symbols are de-
dependent on previous symbols and have considerable structure, i.e. they are very predictable given sufficiently high order conditional probabilities. This leads to less restrictive information sources. In such an information source the occurrence of a source symbol (e.g. an instruction) \( s_i \) may depend upon a finite number \( m \) of preceding symbols. Such a source -- called an \( m \)-th order Markov source -- is specified by the source alphabet \( S \) and the set of conditional probabilities that each symbol \( s_i \) occurs. Even if the probability that a symbol occurs is low, the conditional probability that this symbol occurs could be high.

We may calculate the average information provided by an \( m \)-th order Markov source [1] as follows: if we are in the state specified by \( (s_1, s_2, \ldots, s_m) \), i.e. the \( m \) previous symbols emitted were \( s_1, s_2, \ldots, s_m \), then the conditional probability of receiving symbol \( s_i \) is \( P(s_i|s_1, s_2, \ldots, s_m) \). The information we obtain if \( s_i \) occurs while we are in state \( (s_1, s_2, \ldots, s_m) \) is

\[
I(s_i|s_1, s_2, \ldots, s_m) = -\log_2 P(s_i|s_1, s_2, \ldots, s_m).
\]

Therefore the average amount of information per symbol while we are in state \( (s_1, s_2, \ldots, s_m) \) is given by

\[
H(S) = -\sum_{i=1}^{N} P(s_i) \log_2 P(s_i).
\]

If we average this quantity over the \( 2^m \) possible states, we obtain the average amount of information, or the entropy of the \( m \)-th order Markov source \( S \):

\[
H(S) = -\sum_{i} P(s_i|s_1, s_2, \ldots, s_m) \log_2 P(s_i|s_1, s_2, \ldots, s_m).
\]

Note that if \( S \) is zero memory rather than higher memory, we have the simple fraction

\[
P(s_i|s_1, s_2, \ldots, s_m) = P(s_i).
\]

3 Related works

3.1 Instruction sets

Foster and Gonter [3] describe in their paper a method called conditional interpretation which allows computers to have a large number of instructions without using a large number of bits to hold the opcode. The method is based on redundancy of instruction sequences. They show that if allowing each instruction only seven successors plus one special information about seventy-five percent of the time the desired successor is among the seven permitted successors.

Tanenbaum [12] presents a highly compact instruction encoding scheme, which can reduce program size by factor three using only fixed length opcode and address fields. Tanenbaum proposes a stack-machine, EM-1, with five instruction formats from one byte length up to five byte length, ensuring that the most common statements can be translated into one byte instructions most of the time.

3.2 Entropy

One of the simplest and most common process for reducing the length of a symbol stream is to identify repeated string and replace them with symbols which is described by Wade and Stigall [13]. If one assumes the string \( s_1, s_2, \ldots, s_k \) occurs with probability \( P_k \) and is replaced by the new symbol \( s' \) whenever it occurs, then the new value of entropy \( H' \) is given by

\[
H' = -\frac{P_k}{1 - (k - 1)P_k} \log_2 \left( \frac{P_k}{1 - (k - 1)P_k} \right) - \sum_{i=1}^{k} \frac{P_i - P_k}{1 - (k - 1)P_k} \log_2 \left( \frac{P_i - P_k}{1 - (k - 1)P_k} \right) - \frac{N}{k+1} \frac{P_i - P_k}{1 - (k - 1)P_k} \log_2 \left( \frac{P_i - P_k}{1 - (k - 1)P_k} \right).
\]

The number of symbols to represent a string is reduced by the factor \( i - (k - 1)P_k \). The resulting change in entropy normalized to the length of the original string is given by

\[
H_n = -\left( 1 - (k - 1)P_k \right) H' - H
\]

where \( H_n \) is the normalized entropy.

Hammerstrom and Davidson [4] estimate the information content of a memory referencing stream, using information theory. Specifically, they present techniques for analyzing computer addressing architectures and techniques for analyzing the effectiveness of the addressing architecture and memory/CPU traffic of existing machines with respect to the information theoretic bound for a given trace. To obtain further improvement in memory/CPU bandwidth and CPU addressing efficiency, they suggest looking at higher order memories and accompanying radical changes in CPU architecture and compilation techniques. For the extension to higher order conditional probabilities they define a more involved, but more useful definition of entropy.

Let \( m_j \) be an arbitrary reference to the \( j \)-th location in memory. Let \( g_{j-1}^{n-1} \) be the \( n \) - 1 gram of previous \( n \) - 1 references, where an \( n \) - gram, \( g^n \), is an ordered set of \( n \) addresses. \( G^n \) is the set of all possible \( n \) - grams. The zero order entropy is then

\[
H_0(S) = \log_2 |M|
\]

where \( M \) is the set of memory locations accessed by a program.

The self information of the occurrence of element \( s_i \) is defined as

\[
-\log_2 P(s_i)
\]

and

\[
-\sum_{m_j \in M} \sum_{s_j^{-1} \in G^n} P(m_j, g_{j-1}^{n-1}) \log_2 P(m_j, g_{j-1}^{n-1})
\]

is the \( n \)-th order entropy.
4 Program entropy

The following program entropies were gathered from observations of the SPARC architecture. Sample programs are two C compilers, a debugger, part of a window system, part of the operating system and a statistics program. Statistics about static characteristics of programs are based on more than 300,000 instructions, dynamic characteristics on millions of instructions.

4.1 Static program characteristics

Static characteristics [2, 6, 11] of programs are sufficient for code compaction and easy to collect. A program was written which read the data file and counted the opcodes achieved from the opcode field [10]. All values given in the Table 1 below are average values of all sample programs.

<table>
<thead>
<tr>
<th>order</th>
<th>entropy in bits/opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>static</td>
</tr>
<tr>
<td>0</td>
<td>3.612</td>
</tr>
<tr>
<td>1</td>
<td>2.575</td>
</tr>
<tr>
<td>2</td>
<td>2.202</td>
</tr>
<tr>
<td>3</td>
<td>1.843</td>
</tr>
<tr>
<td>4</td>
<td>1.518</td>
</tr>
<tr>
<td>5</td>
<td>1.198</td>
</tr>
<tr>
<td>10</td>
<td>0.210</td>
</tr>
</tbody>
</table>

Table 1: Entropy of m-th order Markov source.

The average information content or entropy for an m-th order Markov source is shown in Table 1. As we can see there is a decrease of about 29% in entropy for the first order Markov source from the zero order Markov source, about one bit per opcode. For fifth order source we approach an entropy of less than 0.2 bit per opcode and with tenth order we get even less. In fact, there is about 1.5% left of the zero order Markov source.

4.2 Dynamic program characteristics

Dynamic characteristics [8, 12, 14] of programs help in the area of program speed and the overall program run time. They are generally more interesting than static characteristics, but they are also more difficult to collect. Dynamic characteristics of programs were collected by tracing the running program for each instruction thus counting the number of occurrence of each opcode before it is executed.

The average information content for an m-th order Markov source for dynamic programs is shown in Table 1. There is a decrease of about 42% in entropy for the first order Markov source from the zero order Markov source. For the fifth order source we reach an entropy of less than 0.2 bit per opcode and with tenth order we get even less. In fact, there is about 1.5% left of the zero order Markov source.

This shows there is great redundancy in such instruction streams and great dependency between instructions following each other which leads to very high conditional probabilities and therefore to very low entropies. Such low entropies as shown in Table 1 imply we could generate more compact code because the entropy \( H(S) \) gives the number of bits for optimal encoding.

5 Experimental coding results

Our experimental results with arithmetic coding on static and dynamic code are shown in Table 2. The byte stream used for encoding is built with 8 bit long symbols instead of using the structure given in the instruction stream. Therefore the entropy is much higher as seen during opcode measurements, but the results are easily comparable. The upper part of the Table gives the results for dynamic code compression for the zero order Markov model and the first order Markov model. Code compression for the dynamic case means that all bytes which are executed during program run time are compressed. Those same bytes are used to calculate the entropy. In both cases a fixed model is used [15]. The lower part of Table 2 gives our results for static code under the same conditions.

If we compare the compression results with the entropy that can be achieved, the results are remarkable. The compression results match the entropy measured very accurately. Using different streams for different fields from the instruction [10], reflecting the structure given in an instruction stream, we are able to reach even less entropy and therefore encode the instruction stream further.

6 Conclusion

The entropy measurements on SPARC show that we have great opcode dependency in such instruction streams with a lot of redundancy. These opcode dependencies lead to high conditional probabilities. Notably these are the very low entropies for higher order Markov sources for the dynamic case. This, in
particular, is very interesting for problems like memory/CPU bus loading, especially in multi-processor architectures with shared memory. Due to lower entropy in higher order Markov models much more compact encoding is possible for these than for zero order Markov sources. With the right coding technique like arithmetic coding and a sophisticated model taking into account conditional probabilities it is possible to reduce static and dynamic code size considerably. Further more this research outlines a connection between information theory and practical encoding.

References


[8] Patterson, DA and Sequin, CH A VLSI RISC Computer pp. 8–21, September 1982


Table 2: Encoded object code in bytes and bits/symbol with entropy in bits/symbol