©2001 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.″

Optimisation of PPMC Model for Hardware Implementation

C. Feregrino Uribe, S. R. Jones
Loughborough University
{C.Feregrino-Uribe,S.R.Jones}@lboro.ac.uk

Abstract

The development of new and more powerful applications in data communications and computer systems has required an ever-increasing capacity to handle large amounts of data. Lossless data compression techniques have been developed to exploit further available bandwidth of such systems by reducing the amount of data to transmit or store. They have been implemented in both software and hardware. The former approach provides good compression ratios but presents speed limitations. The latter approach offers the possibility of high-speed compression to suit the most demanding applications. Current available hardware implementations are based mainly on LZ (Lempel-Ziv) class of compression schemes. Experience suggests [1] that classical statistical methods, particularly PPM (Prediction by Partial Matching) class of algorithms [2], are impractical for being too slow and resource hungry for hardware realisation. However, there seems to have been relatively little work looking at the potential for re-organising and restructuring the algorithm for hardware implementation. This paper presents a version of the PPMC [3] class of algorithms structured for efficient hardware support and analyses the issues of its hardware implementation.

1. PPMC review

PPM is a state-of-the-art statistical data compression approach [4]. It was originally developed in 1984 and some extensions [5-7] have been proposed since then, including some software implementations [3]. The scheme is based on a system that maintains a dictionary containing a statistical model of the data, assigning probabilities to the symbols and sending these probabilities to arithmetic coder. The probabilities are assigned according to the most recent symbols.

The statistical model in its simplest form counts the number of times each symbol has occurred in the past and assigns a probability to the symbol based on that number. The next higher order model is *context based*, where not just the frequency of the symbol is used to predict but also the frequency a symbol occurred when a particular sequence of symbols immediately preceded that symbol. The preceding symbols are called *context* and the number of them is the *order* of the context.

A PPMC model of order O reads a symbol s and considers the previous O symbols as the current context. Then, it searches for the context followed by the symbol s. If the symbol is not found, the model 'escapes to the next lower order O-1 by transmitting a 'escape code. This process continues until the symbol is found or the model reaches the order 0. If the symbol is not found in order 0, then a final escape is transmitted and the symbol s is predicted by order '-1 , where all symbols have the same probability. The dictionary is then updated adding s to the corresponding contexts.

The following formulas are used to compute symbol and escape probabilities:

$$p(s \mid context) = \frac{f}{t+k} \quad \text{and} \quad p(esc \mid context) = \frac{k}{t+k} \qquad (1)$$

where $p(s|context)$ is the probability that symbol s will occur given that *context* has occurred, f is the frequency count of symbol s, k is the number of different symbols seen in the current context and t is the sum of the frequency counts of all symbols in the current context.

In PPMC model there is a trade-off between compression and speed. Both issues may be exploited individually. If the model exploits compression then it uses a technique called *exclusion* or if it exploits speed it uses *lazy exclusions* [1]. Here we describe the speed model and thus the lazy exclusion technique predicts a symbol only taking into account frequency counts in context levels at or above the context in which it was predicted. Then, when updating the model just these frequency counts are updated.

0-7695-1239-9/01 $10.00 © 2001 IEEE

2. Proposed modifications

This work is mainly focused on simplifying the operations required by the compression model to increase the performance of the whole system. Re-organisation of the algorithm that would benefit greatly the hardware implementation may include:

1) The use of efficient hardware structures to store the dictionary

In compression hardware implementations, efficient structures have been used to store data [9-11]. Among these structures are CAM (Content-Addressable Memory) arrays that allow searching through all the entries in a dictionary simultaneously. The advantage of using this type of structure is that the compression process may be speeded considerably.

2) The limitation of the dictionary size

Practical implementations of any structure must have a limitation in size, particularly taking into account that storage can be expensive in digital technology. Furthermore, space restrictions may warrant that the system fits in a digital device such as FPGA or ASIC. Naturally, severe restrictions in space lead to compression degradation and care must be taken to identify a good trade-off between space and compression.

3) Multi-dictionary model

A CAM array is used to store the dictionary of the model. Recalling that the dictionary contains contexts of different orders, contexts may be stored in a single dictionary or in several ones grouping contexts by order. To implement the former approach, an efficient discard policy has to be implemented to reclaim space once the dictionary is full and before continuing the model adaptation.

The latter approach needs one dictionary per each order of the model. Although it may require more space, it must simplify the discard policy. In this way, a 2nd order model will have 4 dictionaries for -1st, 0th, 1st and 2nd order contexts respectively. Thus, when one of the dictionaries (1st or 2nd order) has no more free space, it can be adjusted, independently of the others.

4) New approaches to update the model

The model updating process requires more than a single modification. Next, we list some proposed modifications we have found to be useful.

a. *Constant total frequency counts*

As mentioned above, the arithmetic coder uses the formulas in (1) to compute symbol and escape probabilities. In these formulas, if the denominator is kept constant and to a power of two, 'divide operations may be replaced by simple 'shifts . Calling this constant denominator '*fixed number of tokens (FNT)*, the formulas would be as follows:

$$p(s\,|\,context) = \frac{f}{FNT} \quad \text{and} \quad p(esc\,|\,context) = \frac{k}{FNT} \quad (2)$$

In (2), by replacing divide operations by shifts, and considering the number of positions to shift the frequencies to the right as a, the integer component of $log_2 FNT$, we get:

$$p(s\,|\,context) = f >> a \quad \text{and} \quad p(esc\,|\,context) = k >> a \quad (3)$$

where symbol '>> represents the 'shift to the right operation.

Additionally, the model may store escape as any other symbol, i.e. with its own frequency counts, to avoid computing its frequency k every time it is needed. In this case, arithmetic coder would require just the first formula in (3) to compute both symbol and escape probabilities.

This proposal should speed up the compression process in the hardware implementation of the model and a different mechanism for assigning frequency counts must be adopted if the compression ratio is to be maintained.

b. *Parallel frequency updating*

Keeping constant the total frequency counts in arithmetic coder is a consequence of the model organisation. In the model, frequency counts should be adjusted in proportion to the constant total and the occurrence of the symbols in the input stream.

We suggest escape to be stored as the first symbol occurring in any context and to assign FNT to that symbol. Later on, we redistribute the tokens among the symbols as they come in. All the symbols should adjust their frequency counts, the quantity of tokens donated by existing symbols to the incoming one should be proportional to the current number of tokens held by each of the existing symbols. The sum of all the donations goes to the incoming symbol.

The operation that allows all symbols to adjust frequency counts according to their occurrence in the input stream is a division by M. Keeping M to a power of

2, a shift operation substitutes the division, then, the number of positions to shift to the right the frequency counts is $m = \log_2 M$. So, symbol j adjusts its frequency by shifting it to the right m positions, $f_j >> m$.

$$f_j = f_j - (f_j >> m) \qquad (4)$$

The result of this operation is a number of tokens that are donated to the incoming symbol. The sum of all these donations is:

$$TIS = \sum_{j=1}^{k} (f_j >> m) \qquad (5)$$

where k is the number of different symbols in the current context.

Looking further into the model implementation, the tokens redistribution operation may be performed in serial or parallel form. The former requires only one shift operator that performs the operations of all k symbols, one at a time, and is time consuming. The second form although requires more operators, it may perform in parallel the shift operations in all frequencies f_j. The modification of the design is focused on increasing compression speed, thus the second form for redistributing tokens is suggested.

c. *Proper identification of updating parameters*

For models with orders higher than 0 there is one component more to consider, the time in which the model stabilises. It is the time when the model has learnt the statistics of the data. Taking as an example a 2nd order model that has contexts of orders −1, 0, 1 and 2, the number of possible contexts in the highest order is too big and just few symbols occur in each context, while in the 0th order context almost all symbols occur. That makes 0th order contexts to have well distributed tokens in their frequencies while in 2nd order contexts escape will have most of the *FNT* tokens. So, since *FNT* and m parameters measure the adjustments in symbol probabilities, they must differ for each order in the model such that closest compression to PPMC is obtained.

d. *Approximation for updating symbol frequencies*

As mentioned in proposal 4b and showed in (5), the model requires adding together the results of the shift operations to collect the tokens for the incoming symbol, *TIS*. Here we propose a new way of computing *TIS* to avoid the time consuming addition.

Since the integer result of the shift operation in (4) is added as in (5), there is no loss of tokens and at any moment of the compression process:

$$FNT = \sum_{j=1}^{k} f_j \qquad (6)$$

A very simple alternative to compute *TIS* is to consider:

$$TIS = FNT >> m \qquad (7)$$

However, using formula (7) to compute the tokens to add to the incoming symbol, *TIS*, showed in some experiments that it generates overflow in arithmetic coder. The reason is that according to formula (7), *TIS* is constant and in order to compute it according to adjustments in PPMC model, the number of different symbols seen, k, must be taken into account.

Thus, to avoid adding together the tokens donated for all the symbols as in formula (4), it is suggested the use of the approximation to the number *TIS* showed in (8).

$$TIS = ((FNT >> m) - 1) - k \qquad (8)$$

The amount *(FNT>>m)-1* is computed only once at the beginning of the compression process and kept constant during the process, we refer to it as *C*, then:

$$TIS = C - k \qquad (9)$$

Formula (9) requires only subtracting k to C per each incoming symbol when obtaining *TIS*. Again, the benefit of this approximation is in compression speed and also in simplifying space requirements.

In summary, to reduce computational complexity and space demands and speed up the PPMC compression process we suggest some modifications to the implementation of the model. They include the replacement of an ever-increasing denominator by a constant one to shift frequencies rather than divide them. And to maintain the accuracy of the model, adjusting frequency counts according to the changes made to the denominator and to the occurrence of the symbols in the input stream.

3. Experimentation results

This section presents the experiments undertaken to study the feasibility of the proposed modifications as well as compression results after applying them to the

algorithm. Previous to the results, some assumptions and methodology are mentioned.

3.1 Assumptions

To simplify the study, the model and coder were separated. The model reads serially the data to be compressed and produces cumulative frequency counts of the symbols that are then transmitted as input to the coder. The arithmetic coder was taken from [12] and adapted to the proposed model. The model is 2nd order, and has 4 dictionaries of 256, 256, 2K and 4K positions for –1st, 0th, 1st and 2nd order contexts respectively. The input stream is compressed in blocks of 4 KB due to this size represents a typical packet size found in many computers and telecommunication systems. Compression is measured as the ratio of output bits and input bits. The model stores escape as any other symbol, i.e., escape has its own frequency counts.

3.2 Methodology

The set of experiments undertaken proves the model operation for a variety of data types over a range of parameter (*FNT* and *m*) values. A C program is used to implement the system and verify its functionality. It used arithmetic coder files from [12]. Additionally, hardware suitability is proved by a 1st order hardware-modelling unit developed under the SystemC [14] modelling platform from the Open SystemC Initiative coupled with arithmetic coder module. We have compiled the system with the VC ++ compiler, version 6.0, in the NT platform.

The data types include the popular Canterbury [13] and Calgary [1] corpus, 'Memory' and 'Thesis' data. Memory data is a selected set of data of about 9 MB contained in memory and includes code and data from the SunOS operating system and 8 real applications and utility programs. A detailed description of the files is given in [9]. Thesis data set is a collection of audio, images, object and text files.

3.3 Experiments

This section contains three main experiments that support our proposals and gives results of them. They are simulations of:

1) A simple 0th order model using a CAM array to store data. It helps to identify the parameters *FNT* and *m* with which the model produces the best compression ratios. It supports proposals 4a and 4b where constant frequency counts and parallel frequency updating is suggested.

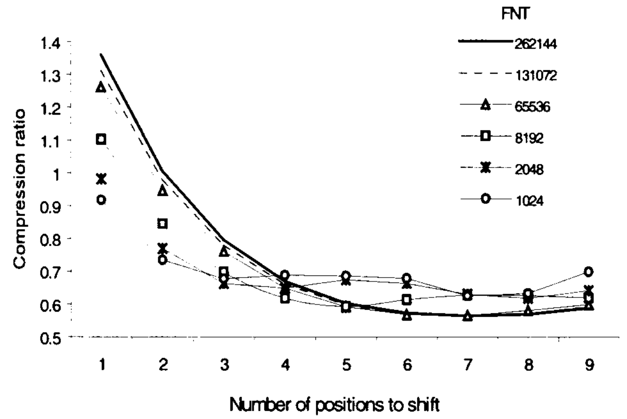

Figure 1. 0th order model compression results as *m* and *FNT* varies

2) Higher order models, 1st and 2nd, verifying results obtained from experiment 1 and identifying new parameters for these models. The proposals 4c and 4d are included in this simulation.

3) All the proposed modifications of the algorithm to improve its hardware implementation.

All experiments include proposals 1, 2 and 3 where the use of efficient hardware structures, limitation of dictionary size and multi-dictionary model were suggested. Specific results of experiments for these proposals are not included in this document due to shortage of space.

Parameter identification, experiment 1

The first experiment consisted on implementing a simple 0th order model using a CAM array. It helps to identify the parameters *FNT*, fixed number of tokens or denominator in formula (1), and *m*, the number of positions to shift frequency counts. It seems helpful to think of *FNT* as a number large enough to be divided among several symbols and of *m* a s a number 'small' enough to help to redistribute the tokens. The task is to find the best numbers for them. We fixed *FNT* and varied *m* and vice versa to find the ones with which the model produced the best compression ratios.

At the beginning of the compression process, *FNT* is assigned to the escape symbol, and later the tokens are redistributed among other symbols as they come in. All

symbols donate tokens to the incoming symbol. Donated tokens are obtained as in formula (4).

Figure 1 shows the trade-off between *FNT* and *m* in a 0th order model. When utilising large *FNT* numbers, there are very good compression ratios (shifting up to 7 positions), close to PPMC. Small *FNT* numbers do not give enough flexibility for the model to update frequency counts thus, poor compression ratios are obtained.

When *m* is small, the model does not reflect symbol probabilities according to their occurrences in the input stream and this fact leads to poor compression.

Higher order models, experiment 2

This experiment consists on implementing models of 0th, 1st and 2nd orders with a wide number of *FNT* and *m* sizes, storing the dictionary in a CAM array and limiting its size as well as approximating the updating symbol frequencies according to formula (8).

Table 1 shows the *FNT* and *m* parameters that exhibit the best compression ratios for the model and they are shown for models of orders 0th to 2nd.

Note that these parameters are selected specially for the higher order models, 1st and 2nd. Each pair *(FNT,m)* in each context order gives the necessary adjustment to proportionate statistics of the data close to PPMC. So, the amount of tokens varies according to the PPMC algorithm, thus providing similar symbol probabilities and compression ratios.

As expected in proposal 4c this table demonstrates how different *FNT* and *m* parameters for distinct contexts orders in a model give better compression ratios than the same parameters for all the context orders. The table shows that a 2nd order model performs well when the fixed number of tokens is 64K for the -1st order, 32K for 0th order, 8K for 1st order and 4K for 2nd order contexts with a number of positions to shift to the right of 7, 6, 4 and 3 respectively.

Using each order different parameters, the arithmetic coder must input the order of the context as well as symbol frequency counts. This is to ensure the proper calculation of symbol probabilities and the correct functionality of the arithmetic coder.

Including all proposals, experiment 3

Next, including all the proposals in last section, the model was simulated in software. Again, 0th, 1st and 2nd order models were tested and several data types were used. Table 2 gives the performance of these models. For each set of data, the first row shows results from PPMC and the second one from our model.

	Model Order	Context Order -1st	0th	1st	2nd
FNT	0th Order	64K	64K		
	1st Order	64K	64K	16K	
	2nd Order	64K	32K	8K	4K
m	0th Order	7	7		
	1st Order	7	7	5	
	2nd Order	7	6	4	3

Table 1. Fixed number of tokens, *FNT,* and number of positions to shift to the right,*m*

Data Set	Model	Order 0th	1st	2nd
Canterbury Corpus	PPMC	0.56	0.45	0.40
	Shift	0.57	0.47	0.43
Memory Data	PPMC	0.61	0.46	0.45
	Shift	0.61	0.49	0.47
Thesis Data	PPMC	0.75	0.67	0.66
	Shift	0.75	0.68	0.66

Table 2. Compression ratio results obtained with PPMC model and the hardware optimised version

Our model performs well compared to PPMC, particularly 0th order model over all data sets and on Thesis data set with all the model orders. A small degradation in compression is observed for Canterbury Corpus and Memory data with 1st and 2nd order models. This is because of the approximation of proposal 4d for updating frequency counts.

4. Analysis

The results from compression ratios show that our model performs very close to PPMC and it is suitable for hardware implementation. Most of the proposed modifications simplify the hardware, which improves compression speed.

The use of tokens for the frequency counts allows parallel updating in the frequencies although it adds some complexity in the hardware design. The redistribution of tokens warrants a proportional adjustment of symbol probabilities according to its occurrence in the input stream.

Approximating the number of tokens to assign to the incoming symbol (*TIS*) yields, in most of the cases, a small degradation in compression ratio. However, it represents great savings in hardware complexity, it changes either an adder tree or a single adder with O(n) delay by a simple subtract operation, as comparing formulas (4) and (8).

5. Hardware modelling

In this section, the architecture of the model is shown. As mentioned above, it was implemented in SystemC [14] platform.

Figure 2 illustrates the architecture of the compressor model, the decompressor has a similar architecture. Arithmetic coder was implemented as a distinct module using the code from [13], which was modified to interact with the model and to accept output signals that the model produces.

Both, compressor and decompressor were designed and the compression results were verified against the C model. As shown in Figure 2, there is one dictionary for each order in the model. The dictionaries have a memory block containing a CAM array and some register arrays for the frequency counts. Its architecture is shown in Figure 3.

A 1st order model has been implemented along with its test bench in SystemC modelling platform from the Open SystemC Initiative (OSCI). This platform allows creating system-level designs in a C++ environment. The addition of one higher order is straightforward.

An estimated of size of the compressor architecture is about 3 million gates. A detailed description is shown in Table 3, where it can be observed that most of the space is assigned to storage and updating of data.

The estimated gate counts in Table 3 are for a 1st order model, which has a dictionary of 4K entries. If one higher order is added to the model, dictionary sizes required would be 2K and 4K for 1st and 2nd order respectively according to some experimental results. This is because most of the matches are generated in the highest context, then less space is required for one lower order. These sizes in the dictionary may be even smaller depending on the data types, however, due to not data is known in advance, we consider the dictionary sizes to be appropriate for any type of data. Thus, the gate account for increasing a second order in the dictionary is increased

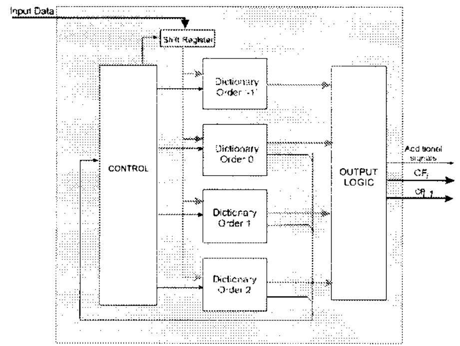

Figure 2. Architecture of the model

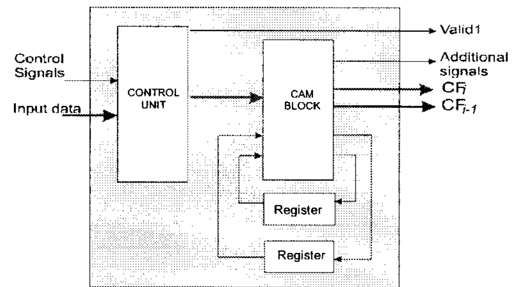

Figure 3. Dictionary architecture

about 1.5 million gates. The decompressor requires the same number of gates as the compressor.

6. Summary and conclusions

A hardware implementation of the statistical PPMC compression model is impractical for the nature of the algorithm. This paper presents a statistical model suitable for hardware implementation as a result of the reorganisation and optimisation of PPMC compression scheme. Some modifications have been proposed as the use of tokens, parallel model updating and substitution of division by shift operations. All the modifications aim to speed up the compression process.

The strategies used and the proposed modifications are by no means exhaustive and other strategies may be used. However, it has been possible to model in hardware the PPMC algorithm and it may be implemented in available digital devices. Using the concept of tokens we have been able to update frequency counts in parallel although not cumulative frequencies. We think that is also possible to update them in parallel and we are currently exploring it.

	Component	Implementation	Estimated gate count
Dictionary Order –1	CAM array	256 x 9 bits registers	18,430
Dictionary Order 0	CAM array	257 x 9 bits registers	34,695
	Frequency table	257 x 16 bits registers	32,895
	Cumulative frequency table	257 x 16 bits registers	32,895
	Shift logic, Adder/subtractor	257 x 16 bits	53,455
Dictionary Order 1	CAM array	4096 x 18 bits registers	1,105,920
	Frequency table	4096 x 16 bits	524,290
	Cumulative frequency table	4096 x 16 bits	524,290
	Shift logic, Adder/subtractor	4096 x 16 bits	851,970
	Mux	4096 x 9 bits	110,590
Additional Logic	Shift register	1 x 18 bits	140
	Control Unit	1 x 4 bits register	30
	Output Logic	1 x 9 bits register	70
	Subtractor	2 x 16 bits	320

Table 3. Estimated size of the compressor architecture

Although the results of proposal 3 are not showed, it is worth to mention that storing contexts in separated dictionaries has the advantage of reducing complexity mainly when dealing with discard policies if other that delete the complete dictionary were used. However, the dictionary storage requirements increase about 20% in the case of a 2nd order model.

Further work involves the synthesis of the SystemC model to directly produce a netlist without translating the model into a HDL language. This saves time by eliminating errors that may be introduced during translation and later may take significant time to track down.

References

[1] T.J. Bell, J.G. Cleary, and I.H. Witten, *Text Compression*, Prentice Hall,Englewood Cliffs, NJ, 1990.

[2] J.G. Cleary and I.H. Witten, "Data Compression using Adaptive Coding and Partial String Matching", IEEE Trans. on Comm., COM-32 (4), April, 1984, pp. 396-402.

[3] A. Moffat, "Implementing the PPM Data Compression Scheme", IEEE Trans. on Comm., COM-38 (11), 1990, pp. 1917–1921.

[4] D. Salomon, *Data Compression. The Complete Reference*, Springer, 1st edition, USA, 1998.

[5] J.G. Cleary and W.J. Teahan, "Unbounded Length Contexts for PPM", Computer Journal Vol. 36 (5), 1993.

[6] P.G. Howard, "The Design and Analysis of Efficient Lossless Data Compression Systems", Report CS-93-28, Department of Computer Sciences, Brown University, Providence, Rhode Island, June, 1993.

[7] C.R. Bloom, "LZP: A New Data Compression Algorithm", IEEE Data Compression Conference, J. Storer editor, Los Alamitos CA, March, 1996, p. 425.

[8] E. Horowitz, S. Sahni, and D. Mehta, *Fundamentals of data structures in C++*, Computer Science Press, United States of America, 1995.

[9] M. Kjelso, M. Gooch, and S. Jones, "The Design and Performance of a Main Memory Hardware Compressor", Proceedings 22nd Euromicro Conference, IEEE Computer Society Press, September, 1996, pp 423-430.

[10] S. Ranganathan and S. Henriques, "High-speed VLSI Design for Lempel-Ziv-based Data Compression", IEEE Trans. Circuits Syst.-II: Analog. Digit. Signal Process, 40(2), February, 1993, pp. 96-106.

[11] C-Y. Lee and R-Y. Yang, "High Throughput Data Compressor Designs using Content-addressable Memory", IEE Proc. Circuits Devices Syst., 142(2), 1995.

[12] I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic Coding for Data Compression", Comm. of the ACM, Vol. 30 (6), June, 1987.

[13] A. Ross and T. Bell, "A Corpus for the Evaluation of Lossless Compression Algorithms", Data Compression Conference, 1997.

[14] SystemC users guide, Version 1.0, Synopsys, Inc., CoWare, Inc. and Frontier Design, Inc., 2000.